THESIS

"FOLLOW THE LEADER" TRACKING BY AUTONOMOUS UNDERWATER VEHICLES (AUVs) USING ACOUSTIC COMMUNICATIONS AND RANGING

by

Daniel P. Kucik

September 2003

Thesis Advisor:	Don Brutzman
Co-Advisor:	Anthony Healey
Co-Advisor:	Doug Horner

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE: "Follow the Leader" Tracking by Autonomous Underwater Vehicles (AUVs) Using Acoustic Communications and Ranging	**5. FUNDING NUMBERS**
6. AUTHOR: Kucik, Daniel P.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAMES AND ADDRESSES Naval Surface Warfare Center – Panama City (NSWC-PC) Office of Naval Research (ONR)	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

ABSTRACT

With advances in computer and sensor technologies, autonomous underwater vehicles (AUVs) are now capable of reaching a level of independent action once thought impossible. Through the use of cooperative behaviors it is possible to further increase their autonomy by allowing multiple operating AUVs to simultaneously coordinate their activities in order to improve the efficiency and effectiveness of the overall system.

This thesis research defines the algorithms and rules needed to perform "follow the leader" cooperative behaviors during AUV rendezvous. This is a low-level first step towards more sophisticated cooperative behaviors, such as swarming or new forms of obstacle/trap avoidance. The approach taken here differs from previous research in that it does not rely on beacons or locator sensors, but instead uses ranging and intention information shared between the vehicles using acoustic communications.

Several tools and algorithms are presented to support the future development of cooperative behaviors. In particular, a previously developed 3D virtual world simulator that utilizes dynamics-based vehicle models has been enhanced to support multiple simultaneously operating vehicles. Finally, a procedural algorithm is shown to correct the relative navigation errors between two vehicles through the use of vehicle-to-vehicle communications and ranging information obtained via acoustic modems.

14. SUBJECT TERMS Autonomous Underwater Vehicle (AUV), Cooperative Behaviors, Acoustic Communications, Follow the Leader, Relative Navigation Error Correction	**15. NUMBER OF PAGES** 208
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

"FOLLOW THE LEADER" TRACKING BY AUTONOMOUS UNDERWATER VEHICLES (AUVs) USING ACOUSTIC COMMUNICATIONS AND RANGING

Daniel P. Kucik
Naval Surface Warfare Center – Panama City
B.S., University of Florida, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED SCIENCE (ROBOTICS)

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Author:	Daniel P. Kucik
Approved by:	Don Brutzman Thesis Advisor
	Anthony J. Healey Co-Advisor
	Douglas P. Horner Co-Advisor
	Don Brutzman Chair, Undersea Warfare Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With advances in computer and sensor technologies, autonomous underwater vehicles (AUVs) are now capable of reaching a level of independent action once thought impossible. Through the use of cooperative behaviors it is possible to further increase their autonomy by allowing multiple operating AUVs to simultaneously coordinate their activities in order to improve the efficiency and effectiveness of the overall system.

This thesis research defines the algorithms and rules needed to perform "follow the leader" cooperative behaviors during AUV rendezvous. This is a low-level first step towards more sophisticated cooperative behaviors, such as swarming or new forms of obstacle/trap avoidance. The approach taken here differs from previous research in that it does not rely on beacons or locator sensors, but instead uses ranging and intention information shared between the vehicles using acoustic communications.

Several tools and algorithms are presented to support the future development of cooperative behaviors. In particular, a previously developed 3D virtual world simulator that utilizes dynamics-based vehicle models has been enhanced to support multiple simultaneously operating vehicles. Finally, a procedural algorithm is shown to correct the relative navigation errors between two vehicles through the use of vehicle-to-vehicle communications and ranging information obtained via acoustic modems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION **1**
- **A. PROBLEM STATEMENT** **1**
- **B. OVERVIEW** **2**
- **C. MOTIVATION** **3**
- **D. OBJECTIVES** **3**
- **E. THESIS ORGANIZATION** **4**

II. BACKGROUND AND RELATED WORK **5**
- **A. INTRODUCTION** **5**
- **B. AUTONOMOUS UNDERWATER VEHICLES (AUVs)** **5**
  - **1. Overview** **5**
  - **2. Acoustic Radio Interactive Exploratory Server (ARIES)** **5**
  - **3. Remote Environmental Measurement UnitS (REMUS)** **8**
  - **4. AUV Navigation** **8**
    - ***a. Overview*** ***8***
    - ***b. Dead Reckoning (DR)*** ***9***
    - ***c. Acoustic Long Baseline (LBL)*** ***10***
    - ***d. Global Positioning System (GPS)*** ***11***
- **C. ACOUSTIC COMMUNICATIONS (ACOMMS)** **13**
  - **1. Overview** **13**
  - **2. Ranging Function** **14**
  - **3. Limitations** **15**
- **D. COORDINATE SYSTEM DEFINITIONS** **17**
  - **1. Overview** **17**
  - **2. Global Reference Frame** **18**
  - **3. Body Fixed Frame of Reference** **19**
- **E. RELATIVE NAVIGATION ERROR CORRECTION** **20**
- **F. AUV WORKBENCH SIMULATOR** **22**
  - **1. Overview** **22**
  - **2. Supporting Technologies** **22**
    - ***a. Overview*** ***22***
    - ***b. Extensible Markup Language (XML)*** ***22***
    - ***c. Extensible Style Sheet Language Transformations (XSLT)*** ***22***
    - ***d. Simple API for XML (SAX)*** ***23***
    - ***e. Virtual Reality Modeling Language (VRML)*** ***23***
    - ***f. Extensible 3D (X3D) Graphics*** ***24***
    - ***g. Extensible Java 3D (Xj3D) Browser*** ***25***
  - **3. AUV Workbench** **26**
  - **4. History and Contributors** **27**
- **G. SUMMARY** **29**

III. TRIANGULATION THEORY: GEOMETRY AND EQUATIONS 31
A. INTRODUCTION..31
B. TRIANGULATION USING TWO RANGE VALUES 31
1. Overview 31
2. Assumptions...31
3. Triangulation Theory: Geometry and Equations 31
a. Overview *31*
b. System Configuration and the Triangulation Reference Frame *32*
c. Triangulation Equations *33*
4. Conversion to the Global Reference Frame 36
a. Overview *36*
b. Triangulation Frame with Transponder 1 as the Origin *36*
c. Triangulation Frame with Transponder 2 as the Origin *38*
C. TRIANGULATION USING THREE OR MORE TRANSPONDERS 40
D. OPTIMAL TRIANGULATION GEOMETRY AND OBSERVABILITY 41
E. SUMMARY 43

IV. RELATIVE NAVIGATION ERROR CORRECTION 45
A. INTRODUCTION..45
B. FOLLOW-THE-LEADER TACTICS 45
1. Overview 45
2. Track/Path Following 45
3. Relaxed Following..46
C. PROBLEM STATEMENT 47
D. OBJECTIVES 48
E. APPLICATIONS 50
F. PROPOSED SOLUTION..50
1. Overview 50
2. Assumptions...51
3. Range Data Collection 51
a. Overview *51*
b. Preprogrammed Test Points (TPs) *52*
c. Dynamically Determined Test Points (TPs) *55*
d. Test Point (TP) Configurations *55*
4. Relative Navigation Error Correction Factor Calculation 63
a. Overview *63*
b. Correction Factor Calculation Using Two Points.................*63*
c. Correction Factor Calculation Using Three or More Test Points *67*
G. SUMMARY 68

V. MATLAB EVALUATION OF THE RELATIVE-NAVIGATION ERROR-CORRECTION ALGORITHM 69
A. INTRODUCTION..69
B. OBJECTIVES 69

C. ASSUMPTIONS 69
D. APPROACH 69
E. ANALYSIS 72
1. **Overview** 72
2. **Varying the Time Required to Complete the Correction Procedure** 72
3. **Varying Baseline Length** 74
4. **Varying Baseline Distance to the LV** 76
5. **Effects of Random Ranging Errors** 79
6. **Single Run Results Using Optimal Settings** 80
F. SUMMARY 83

VI. "FOLLOW-THE-LEADER" COOPERATIVE BEHAVIOR 85
A. INTRODUCTION 85
B. PROBLEM STATEMENT 85
C. OBJECTIVES 86
D. PROPOSED SOLUTION 86
1. **Overview** 86
2. **Assumptions** 86
3. **Approach** 87
4. **Follow-the-Leader Behavior** 87
a. Overview 87
b. FV Waypoint Calculation 87
c. Calculation of the LV's Transit Speed 91
d. Follow-the-Leader Behavior Execution 91
E. SUMMARY 94

VII. AUV WORKBENCH SIMULATOR 95
A. INTRODUCTION 95
B. MODIFICATIONS AND IMPROVEMENTS 95
1. **Overview** 95
2. **Xj3D Browser Upgrade** 95
3. **Support for Multiple Heterogeneous Vehicles** 95
4. **Report Generation** 96
5. **Vehicle-to-Vehicle Communications** 97
C. IMPLEMENTATION OF "FOLLOW THE LEADER" 97
D. SIMULATION RESULTS 100
1. **Overview** 100
2. **Performance with Perfect Communications** 100
3. **Performance with Communications Drop-Outs** 102
E. SUMMARY 104

VIII. CONCLUSIONS AND RECOMMENDATIONS 105
A. INTRODUCTION 105
B. GENERAL THESIS CONCLUSIONS 105
C. SPECIFIC CONCLUSIONS 105
1. **Overview** 105

2. Relative-Navigation Error-Correction Behavior 105
3. "Follow-the-Leader" 106
4. AUV Workbench 106
D. RECOMMENDED FUTURE WORK 106
1. Overview 106
2. Relative-Navigation Error-Correction Procedure and Matlab Simulation 106
3. Follow-the-Leader Cooperative Behavior and the AUV Workbench 107
4. AUV Workbench 107

APPENDIX A. ACRONYMS AND ABBREVIATIONS 109

APPENDIX B. MATLAB CODE FOR THE RELATIVE-NAVIGATION ERROR-CORRECTION SIMULATION 111
A. INTRODUCTION 111
B. GRAPHICAL USER INTERFACE (GUI) 111
1. Overview 111
2. Graphical User Interface (GUI) Handles 111
C. GUI DATA PROCESSING 114
1. Overview 114
2. DataEntryGUI.m 114
D. PRIMARY PROCESSING 123
1. Overview 123
2. CorrectNavErrors.m 123
E. CORRECTION FACTOR CALCULATION USING TWO RANGE VALUES 132
1. Overview 132
2. SimpleCorrectionFactorCalc.m 132
F. CORRECTION FACTOR CALCULATION USING THREE OR MORE RANGE VALUES 134
1. Overview 134
2. SOECorrectionFactorCalc.m 134
G. LV TRACK CALCULATION 135
1. Overview 135
2. CalculateLVTrack.m 135
H. FV TRACK CALCULATION 137
1. Overview 137
2. CalculateFVTrack.m 137
I. CORRECTION FACTOR ANALYSIS 139
1. Overview 139
2. AnalyzeCorrectionFactor.m 139
J. SIMULATION RESULTS OUTPUT FILE 140
1. Overview 140
2. OutputDataToFile.m 140
K. CORRECTION FACTOR PERFORMANCE TRACK GENERATION 142

1. **Overview** 142
2. **GenerateComparisionTracks.m** 142

L. GRAPH GENERATION 144

1. **Overview** 144
2. **GenerateSummaryTrackPlot.m** 144
3. **GenerateSummaryTPPlot.m** 145
4. **GenerateSummaryResultsPlot.m** 146

M. MISCELLANOUS SUPPORT FUNCTIONS 147

1. **Overview** 147
2. **Range.m** 147
3. **CalculateRanges.m** 147
4. **RotateRefFrame.m** 148
5. **Deg2rad.m** 149
6. **Rad2deg.m** 149
7. **DetermineTPConfigForParallel.m** 149

APPENDIX C. AUV WORKBENCH CODE 151

A. INTRODUCTION 151

B. SAMPLE VRML FILE 151

1. **Overview** 151
2. **GeneralAUVMissionScenerio.wrl** 151

C. JAVA / VRML INTERFACE 154

1. **Overview** 154
2. **AUVVRMLController.java** 155

D. ARIES DYNAMICS AND IMPLEMENTATION OF "FOLLOW-THE-LEADER" 159

1. **Overview** 159
2. **AriesDynamics.java** 159

LIST OF REFERENCES 177

INITIAL DISTRIBUTION LIST 183

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Common divide-and-conquer approach for multiple AUV minefield-search missions. 2
Figure 2. Shipboard handling of NPS' Acoustic Radio Interactive Exploratory Server (ARIES) AUV aboard the Portuguese research vessel ARGUIPELIGO during Azores operations in August 2001 (from NPS Center for AUV Research, 2003b)....6
Figure 3. ARIES hardware schematic showing all primary systems and components (from Healey 2001).....7
Figure 4. Hydorid's Remote Environmental Measurement UnitS (REMUS) AUV is two-man portable and does not require specialized equipment or a crane for deployment or recovery (from Hydroid 2003).....8
Figure 5. Typical accumulated dead-reckoning (DR) errors as a function of time for error rates ranging from 1.5 to 3 percent distance traveled (%DT) (without LBL or GPS position updates).....10
Figure 6. Acoustic long-baseline navigation systems use range readings to triangulate the vehicle's position. Vehicles typically operate on only one side of the x-axis to eliminate the ambiguity resulting from the range rings intersecting in two locations. 11
Figure 7. GPS uses pseudoranges and ephemeris information to triangulate the positions of receivers located on or near the surface of the earth (from University of Pennsylvania NROTC 2003).....12
Figure 8. NPS' Acoustic Radio Interactive Exploratory Server (ARIES) AUV performing a GPS pop-up during Azores operations in August 2001 (from Nguyen 2002)....13
Figure 9. The basic operation and data flow for typical acoustic modems.....14
Figure 10. Benthos 89x acoustic modem performance in 15m of water with the transducers placed at a depth of 8m. (from Marr 2003) 16
Figure 11. Benthos 89x acoustic modems performance with a diverging channel where the data is sent from approximately 15m of water to a modem in approximately 30m of water with both modem transducers placed at a depth of 8m. (from Marr 2003)....16
Figure 12. Benthos 89x acoustic modems performance with a converging channel where the data is sent from approximately 30m of water to a modem in approximately 15m of water with both modem transducers placed at a depth of 8m. (from Marr 2003)....17
Figure 13. The global / NED reference frame has some origin O located at the surface of the water and axes aligned in the north, east, and down directions....18
Figure 14. Angles measured in the horizontal plane of the global / NED reference frame, such as heading and course, are referenced to the north axis, such that 0-degrees represents north, 90-degrees is east, 180-degrees is south, and 270-degrees is west. 19

Figure 15. Starboard view of the body fixed reference frame with origin O' and the y-axis coming out of the page. 20
Figure 16. Stern view of the body fixed reference frame with origin O' and the x-axis going into the page. 20
Figure 17. XML example demonstrating the use of tags to store name information. 22
Figure 18. VRML example defining a 1m x 1m x 1m white square. 23
Figure 19. Rendering of the 1m x 1m x 1m white square defined in Figure 18 using Internet Explorer and the Cortona VRML plug-in. 24
Figure 20. X3D equivalent of the 1m x 1m x 1m white box defined in Figure 18. 25
Figure 21. Xj3D rendering of the X3D 1m x 1m x 1m white square defined in Figure 20. 26
Figure 22. The graphical user interface (GUI) for the original AUV Workbench (from Grunesien 2002). 27
Figure 23. Mission planning version of the AUV Workbench showing ARIES using its obstacle avoidance algorithm to maneuver around Manta mines. 28
Figure 24. The triangulation reference frame where the origin is located at transponder 1, the x-axis is aligned with the baseline and runs through transponder 2, and the y-axis is perpendicular to the baseline and is positive in the direction of the work area. 32
Figure 25. Baseline crossings result in ambiguities because the range rings centered on the transponders intersect in two places, so the LBL system is not able to determine if the vehicle is located at Position 1 or Position 2. 33
Figure 26. Range readings (R_{T1} and R_{T2}) obtained from the transponders represented as range rings centered on the transponders. 34
Figure 27. A simplified interpretation of the range readings after the exclusion of the secondary intersection of the range rings located on the "wrong" side of the baseline. 34
Figure 28. Simplified triangulation computation using the x and y values of the vehicle's position. 35
Figure 29. Triangulation frame's orientation relative to the NED frame with transponder 1 as the origin of the triangulation frame. 37
Figure 30. Triangulation frame's orientation relative to the NED frame with transponder 2 as the origin of the triangulation frame. 39
Figure 31. Position fixes calculated using triangulation techniques with three or more transponder can be represented as the intersection of the range rings centered on the transponders. 41
Figure 32. Optimal vehicle operating area for a two transponder LBL system, which ensures the triangulation solution is observable (Matos 1999). 42
Figure 33. Track/path following operations where the following vehicle matches the precise track taken by the leader. 46
Figure 34. Relaxed following requires the FV to stay within a given min/max range from the LV while loosely following the LV's track. 46
Figure 35. Two vehicles performing track/path following based on position estimates obtained using vehicle-to-vehicle communications will result in a total track following error equal to the sum of the errors for the two vehicles. 48

Figure 36. Test point programming for the parallel track correction configuration where the FV’s track is parallel to the LV’s track. 53
Figure 37. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the port side of the vehicle and TP1 is behind the LV with TP2 as the origin of the reference frame. 56
Figure 38. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the port side of the vehicle and TP1 is ahead of the LV with TP2 as the origin of the reference frame. 57
Figure 39. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the starboard side of the vehicle and TP1 is behind the LV with TP2 as the origin of the reference frame with TP2 as the origin of the reference frame. 58
Figure 40. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the starboard side of the vehicle and TP1 is ahead of the LV with TP2 as the origin of the reference frame. 59
Figure 41. Triangulation reference frame for the follow-the-leader test point configuration where TP1 is located on the starboard side of the LV with TP2 as the origin of the reference frame. 60
Figure 42. Triangulation reference frame for the follow-the-leader test point configuration where TP1 is located on the port side of the LV with TP2 as the origin of the reference frame. 60
Figure 43. Relative motion plot showing the FV’s behavior during the execution of the parallel track TP configuration, corresponding to the maneuver geometry defined in Figure 37. 61
Figure 44. Relative motion plot showing the FV’s behavior during the execution of the follow-the-leader TP configuration, corresponding to the maneuver geometry depicted in Figure 41. 62
Figure 45. The graphical user interface (GUI) for the Matlab-based relative navigation error correction simulation. 70
Figure 46. The test point configuration for evaluating the effects of relative errors accumulated during the relative-navigation error-correction procedure. 73
Figure 47. The results of varying the time required to complete the correction procedure with a baseline length of 200m, the baseline located 200m behind the LV, DR time of 1800 seconds before the start of the correction behavior, and compass biases determined by the automatic data collection function. 74
Figure 48. The test point configuration for the varying baseline length experiment where the vehicle is located 200m from the baseline and the length of the baseline is varied. 75
Figure 49. The results of varying the baseline length with the baseline located 200m behind the LV, DR time of 1800 seconds before the start of the correction behavior, the time to complete the correction procedure based on the distance between the test points, and the data was collected using the automatic data collection function. 76

Figure 50. The test point configuration for the varying LV distance from baseline experiment where the baseline length remains fixed at 200m and the LV's distance from the baseline is varied. 77
Figure 51. The results of varying the LV's distance to the baseline with a baseline length of 200m, and DR time of 1800 seconds before the correction. 78
Figure 52. The impact of ranging errors on the navigation error correction algorithm's performance with a baseline length of 200m, a distance of 200m between the LV and baseline, the probability of ranging error set to 0.4, the maximum ranging error varied between 0 and 20, and compass biases determined by the automatic data collection function. 79
Figure 53. Test point configuration for the single run simulation using optimal settings. 80
Figure 54. The true and estimated tracks for the LV and the FV before and during the correction procedure. Actual motion by the FV does not cross the LV's track as expected by DR navigation....81
Figure 55. The LV and FV tracks during the correction procedure and the positions of the vehicles at the time each TP is reached (indicated by O). Actual motion by the FV does not cross the LV's track as expected by DR navigation....82
Figure 56. The results of the correction where the FV attempts to perform track following with the LV at a distance 100m behind the LV (note that DR errors continue to accumulate during the track-following behavior). The expected relative motion necessary for track following is achieved. 83
Figure 57. The position of the FV's waypoint is calculated using the position of the LV's current waypoint and its course. Both the LV and the FV share the same track in this example. 88
Figure 58. When using Equation 27 to determine the distance of the carrot from the LV's waypoint, the worst-case minimum distance between the vehicles is d_{min} and the maximum distance is the sum of the diameters for each position regions and d_{min}. 90
Figure 59. Flowchart of the FV's logic for the follow-the-leader behavior. 92
Figure 60. Flowchart of the LV's logic for the follow-the-leader behavior....93
Figure 61. Data flow diagram and module structure for the AUV Workbench with support for multiple vehicles. 96
Figure 62. The AUV Workbench user interface and 3D virtual world rendering of REMUS and ARIES performing the follow-the-leader cooperative behavior....98
Figure 63. The AUV Workbench user interface and 3D virtual world rendering of REMUS and ARIES performing the follow-the-leader cooperative behavior when a communications drop-out occurs, which results in ARIES entering a loiter behavior while REMUS continues its raster minefield search. 99
Figure 64. The intended track of the LV during test and evaluation of the follow-the-leader behavior performed using the AUV Workbench. 100

Figure 65. LV and FV tracks during the execution of the follow-the-leader cooperative behavior with perfect communications. 101
Figure 66. LV and FV waypoints during the execution of the follow-the-leader cooperative behavior with perfect communications. 102
Figure 67. LV and FV tracks during the execution of the follow-the-leader cooperative behavior with a 60 second communications drop-out at the end of the fourth leg. 103
Figure 68. LV and FV waypoints during the execution of the follow-the-leader cooperative behavior with a 60 second communications drop-out at the end of the fourth leg. 104
Figure 69. Relative-navigation error-correction Matlab graphical user interface (GUI). 112

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1 A description of the GUI fields for the Matlab simulation used to evaluate the relative navigation error correction algorithm. 71
Table 2 Acronyms and abbreviations...109
Table 3 The graphical user interface (GUI) handles for the Matlab relative-navigation error-correction computations. 113

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank the following individuals and organizations for their support.

1. The Office of Naval Research (ONR) and the Naval Surface Warfare Center – Panama City (NSWC-PC) for sponsoring my studies at the Naval Postgraduate School (NPS).
2. Professor Don Brutzman, Professor Anthony Healey, and Doug Horner for their guidance, support, and encouragement.
3. RADM John Pearson, USN (retired) for securing the funding which made my studies at NPS and this thesis possible. I would also like to thank RADM Pearson for his support and guidance over the past year.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

With advances in computer and sensor technologies, autonomous underwater vehicles (AUVs) are now capable of performing tasks that were once thought impossible. Currently there are research efforts underway to develop new methods to improve the efficiency and effectiveness of operations completed using AUVs. One such area of research is cooperative behaviors, which enables multiple simultaneously operating AUVs to coordinate their efforts to meet the mission objectives.

This thesis research defines the algorithms and rules needed to perform the low-level cooperative behavior of “follow-the-leader” during AUV rendezvous, which provides the tools needed to develop more sophisticated cooperative behaviors, such as swarming and group-level obstacle/trap avoidance. This thesis explores two key components needed to successfully implement the follower-the-leader behavior. The first is to minimize the relative navigation error between vehicles that do not share a common long-baseline navigation (LBL) system. This step is necessary because it reduces the likelihood of vehicle-to-vehicle collisions and increases the accuracy of the vehicle-following behavior. The second component is the follow-the-leader or pursuit algorithm and the rules required to cope with the somewhat unpredictable nature of the acoustic communications channel.

Both the relative-navigation error-correction procedure and the follow-the-leader algorithm utilize vehicle-to-vehicle communications and ranging information obtained using commercial-off-the-shelf (COTS) acoustic modems. Acoustic modems provide limited bandwidth and their reliability is range dependent, so to provide a robust solution the proposed algorithms were designed to cope with these limitations. Furthermore, a distributed control structure was selected to minimize the communications requirements and to further increase the robustness of the system.

B. OVERVIEW

The U.S. Navy and other governmental agencies are currently using or considering the use of AUVs for a wide variety of applications, such as force protection, reconnaissance, mine countermeasures (MCM), and oceanography. Hydroid's Remote Environmental Measurement UnitS (REMUS) AUV was introduced to the fleet on a limited basis several years ago and it was recently used by the Navy during Operation Iraqi Freedom to support MCM operations in the port of Umm Qasr (Coleman 2003).

It is not uncommon to have multiple vehicles simultaneously operating in the same work area to reduce the time required to complete a mission. In most cases a simple divide-and-conquer approach is taken, such as depicted in Figure 1, in which the primary work area is divided and each vehicle operates independent of the other vehicles in its own sub-area. This approach requires all vehicles to have a full sensor suite and provides only a linear reduction in the time required to complete the mission.

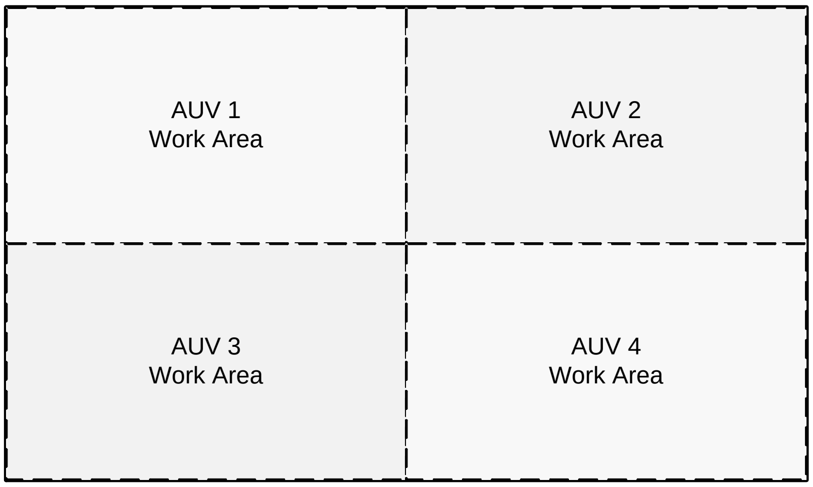


Figure 1. Common divide-and-conquer approach for multiple AUV minefield-search missions.

With the use of cooperative behaviors and the sharing of near real-time information between vehicles, it is possible for simultaneously operating AUVs to coordinate their efforts. Cooperative behaviors allow vehicles to adjust their actions and respond to information collected by other vehicles in the group, thereby allowing the vehicles to dynamically adjust their actions to better cope with the situation at hand. Furthermore, this approach has the potential of providing a better-than-linear reduction of the required mission time and also allows for additional vehicle configuration flexibility.

C. MOTIVATION

There are two primary motivating factors for this research. The first is to develop the basic foundation and tools needed to develop sophisticated cooperative behaviors for AUVs. Through the use of such cooperative behaviors it will be possible to increase the overall level of autonomy and to improve efficiency and the effectiveness of autonomous systems.

The second motivation is to support the current research efforts at NPS involving the Acoustic Radio Interactive Exploratory Server (ARIES) AUV. A future objective for ARIES is to have it collect data from one or more AUVs through an acoustic modem and then relay the information over a radio frequency (RF) link to other vehicles or to an operator not within acoustic communications range. Since the maximum baud rate for acoustic modems is achieved when the distance between the sending and receiving modems is minimized, the “follow the leader” behavior will allow ARIES to rendezvous and closely follow other AUVs to perform in-transit high-speed data transfers, thereby allowing multiple-AUV searches to proceed for long durations.

D. OBJECTIVES

This thesis defines the algorithms and rules needed to implement a “follow-the-leader” cooperative behavior in AUVs. To successfully meet this goal and to provide the tools needed for the future development of cooperative behaviors, the following sub-tasks must be completed.

1. Develop the procedure and algorithms needed for reducing the relative navigation errors between vehicles that do not use GPS or share a common long baseline navigation system.

2. Develop the follow-the-leader algorithm and rules necessary to cope with the somewhat-unpredictable nature of acoustic communications.

3. Develop reusable software tools that can be leveraged to develop future cooperative behaviors.

4. Validate the developed algorithms using physics-based modeling and simulation (M&S).

5. Future work is to test the algorithms in the water using two or more AUVs.

E. THESIS ORGANIZATION

Chapter II discusses similar research and provides general background information to enhance the reader's understanding of concepts developed in this thesis. Chapter III explains the theory and equations used for triangulating positions using ranging information. Chapter IV examines the relative navigation error problem and proposes a solution for correcting these errors based on the concepts presented in Chapter III. Chapter V discusses the Matlab simulator used to evaluate the relative navigation error correction algorithm and presents the results. Chapter VI explores the follow-the-leader problem and proposes a set of algorithms and rules. Chapter VII discusses the AUV Workbench and its use for simulating the follow-the-leader algorithm. Chapter VIII provides thesis conclusions and recommendations for future work.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter provides background information and summarizes related research to aid the reader's understanding of material presented in later chapters.

B. AUTONOMOUS UNDERWATER VEHICLES (AUVs)

1. Overview

An autonomous underwater vehicle (AUV) is an unmanned, free-swimming vehicle that requires little or no human intervention after it is deployed. Most AUVs resemble miniature submarines or torpedoes that move through the water using a propulsion system typically consisting of electric motors and propellers. Data collected from inertial sensors, an acoustic Doppler velocity log (DVL), compass, acoustic long baseline (LBL), and/or other navigation sensors is integrated and processed by an onboard computer that typically uses a Kalman filter to estimate the position and orientation of the vehicle. The overall control and behavior of the vehicle is controlled by an on-board computer, which enables the vehicle to complete a mission without the need for low-level operator commands. AUVs are typically battery powered. In some cases more powerful energy sources are used, such as fuel cells (Tervalon 2003).

In addition to navigation sensors, AUVs typically carry additional sensors to collect environmental and other types of information. There are many application specific sensors, but some common sensors are conductivity, temperature, and depth (CTD) sensors, side-scan sonar to search for objects on the sea floor, forward-looking sonar for obstacle avoidance, and acoustic Doppler current profilers (ADCPs) for monitoring water currents.

2. Acoustic Radio Interactive Exploratory Server (ARIES)

The ARIES AUV was developed and constructed by the Naval Postgraduate School (NPS) Center for AUV Research. ARIES is designed to function as a communications and data server to collect and distribute data between multiple vehicles and/or the system operator. For example, ARIES might collect data from other AUVs operating in the work area and then surface to transmit the collected data to the operator via a radio frequency (RF) modem. In addition to serving as a communications server, ARIES is easily modified to support a wide variety of research topics and has been the subject of numerous theses and dissertations.

ARIES is 10”x16”x120” (25 x 40 x 304 cm), 490 pounds (220 kg), and it has a maximum speed of 4 knots (2 m/s). Onboard sensor include an acoustic Doppler current profiler (ADCP) for monitoring water currents and a video camera for identifying mine-like objects on the sea floor (NPS Center for AUV Research, 2003a). The vehicle’s navigation system include the above-mentioned ADCP which acts as a Doppler velocity log (DVL), an inertial measurement unit (IMU), compass, and a Kalman filter is used to combine the navigation sensor data to calculate an estimate of the vehicle’s position and orientation. ARIES is also equipped with a Differential Global Positioning System (DGPS) receiver, which corrects the dead-reckoning (DR) solution when the vehicle surfaces.

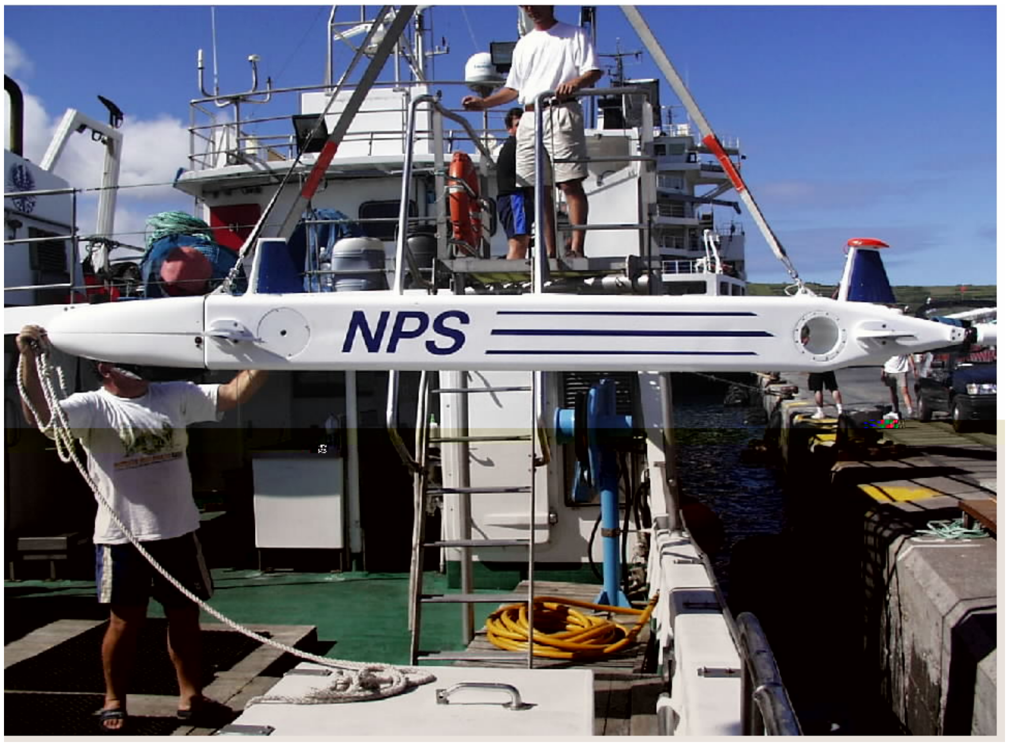


Figure 2. Shipboard handling of NPS’ Acoustic Radio Interactive Exploratory Server (ARIES) AUV aboard the Portuguese research vessel ARGUIPELIGO during Azores operations in August 2001 (from NPS Center for AUV Research, 2003b).

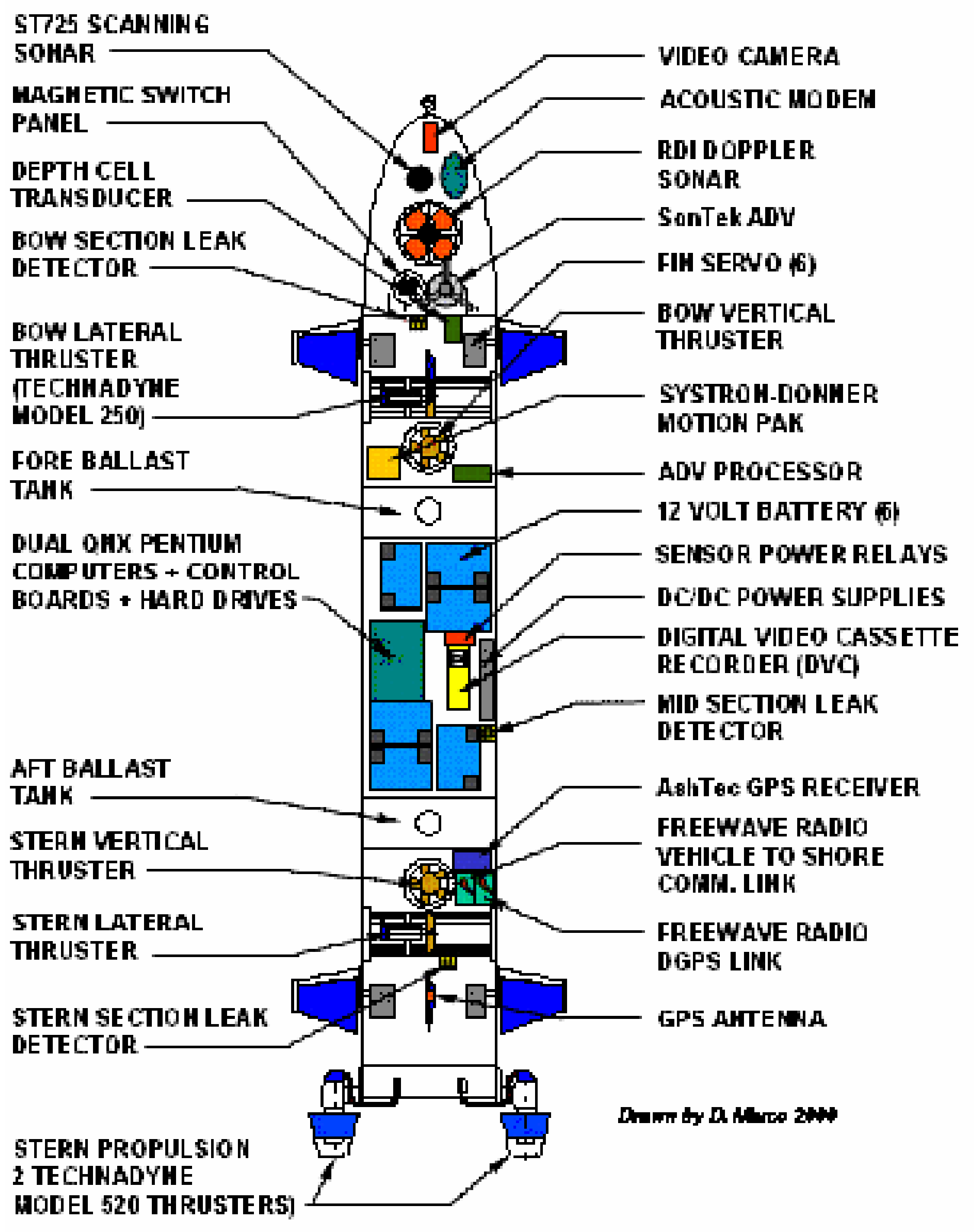


Figure 3. ARIES hardware schematic showing all primary systems and components (from Healey 2001).

3. Remote Environmental Measurement UnitS (REMUS)

The original REMUS AUV was designed and built as a research vehicle at the Woods Hole Oceanographic Institute (WHOI). In 2001 REMUS entered commercial production and is now sold by Hydroid, Inc. REMUS is used for a number of applications including oceanographic surveying and mine countermeasures.

REMUS is a two-man portable system that is 7.48” (19 cm) in diameter, 63” (160 cm) long, and weighs 80 pounds (37 kg). There are a number of sensor options for REMUS, such as side-scan sonar, an ADCP, light scattering sensors, CTD sensors, and the Dual Frequency Identification Sonar (DIDSON) (Belcher 2003) and (Hydroid 2003). The navigation sensor suite includes a compass, the above-mentioned ADCP to provide speed over ground when ground lock is available, and an acoustic long-baseline (LBL) system or an optional GPS receiver to correct accumulated DR errors.

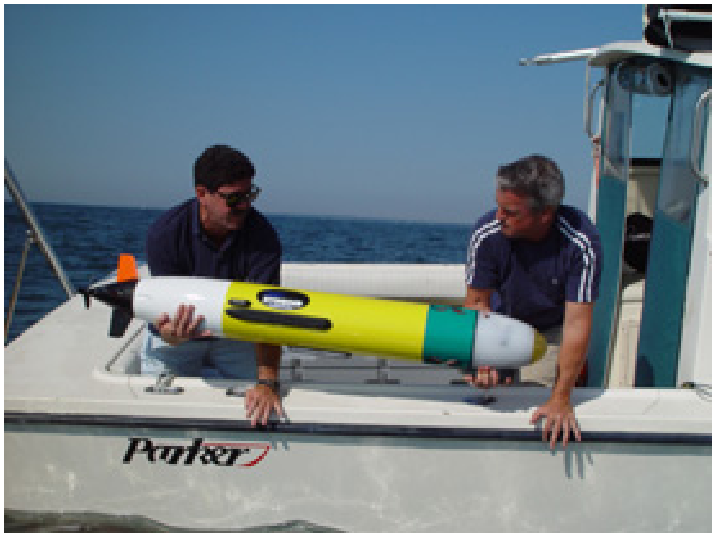


Figure 4. Hydorid’s Remote Environmental Measurement UnitS (REMUS) AUV is two-man portable and does not require specialized equipment or a crane for deployment or recovery (from Hydroid 2003).

4. AUV Navigation

a. Overview

To successfully complete most missions it is essential that AUVs can accurately estimate their position. For example if an AUV is used to perform a search / classify / map (SCM) operation for mine countermeasures (MCM), it is critical that the vehicle be able to accurately determine the position of detected mine-like objects to later enable efficient reacquire / identify / neutralize (RIN) operations. In the case of oceanographic surveys, the vehicle must accurately know its position or else the collected data and mapping information will be inaccurate.

An AUV navigation system typically consists of two components. The first component is dead reckoning (DR), which uses internally mounted sensors to estimate the orientation and position of the vehicle based on estimated course and speed over ground. The second component of a navigation system utilizes sensors like an acoustic long baseline (LBL) or the Global Positioning System (GPS) to provide periodic position updates to minimize the accumulated navigation errors associated with dead reckoning.

b. Dead Reckoning (DR)

Dead-reckoning navigation utilizes sensors internal to the vehicle, such as inertial measurement units (IMUs), Doppler velocity logs (DVLs), compasses, Hall Effect sensors to monitor propeller speed, and/or other similar sensors. The data from these sensors is typically combined using a Kalman filter to cumulatively estimate the vehicle's position and orientation. Accumulated errors associated with DR are often significant and are usually on the order of one to three percent of the distance traveled (%DT), though some newer integrated IMU/DVL systems advertise as little as 0.05 %DT (Kearfott 2001). For ARIES, under minimal set and drift current conditions, the dead reckoning errors were found to be between 2.22 and 2.72 %DT without the use of inertial sensors (Nguyen 2003). DR errors are a result of compass bias, inertial sensor drift, etc., which can lead to heading errors as large as 5-degrees (Alleyne 2000). The accumulative effect of these errors over time is shown in Figure 5.

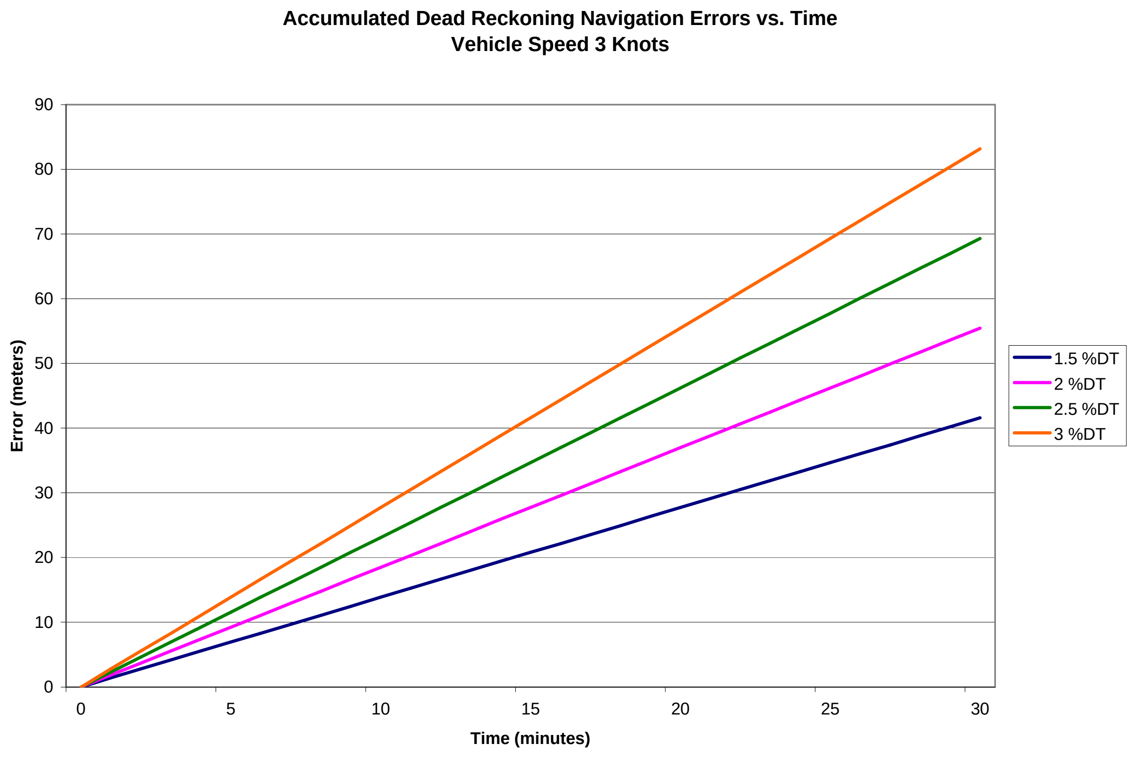


Figure 5. Typical accumulated dead-reckoning (DR) errors as a function of time for error rates ranging from 1.5 to 3 percent distance traveled (%DT) (without LBL or GPS position updates).

c. Acoustic Long Baseline (LBL)

Long baseline (LBL) navigation systems provide periodic position updates to supplement the dead reckoning solution to minimize accumulated DR navigation errors. LBL systems consist of at least two transducers installed at “surveyed” positions and a receiver installed in the vehicle. The vehicle determines its range from each transponder by measuring the amount of time it takes acoustic pressure waves or “pings” sent by the transponders to reach the vehicle. The calculated ranges are then used to triangulate the position of the vehicle as shown in Figure 6.

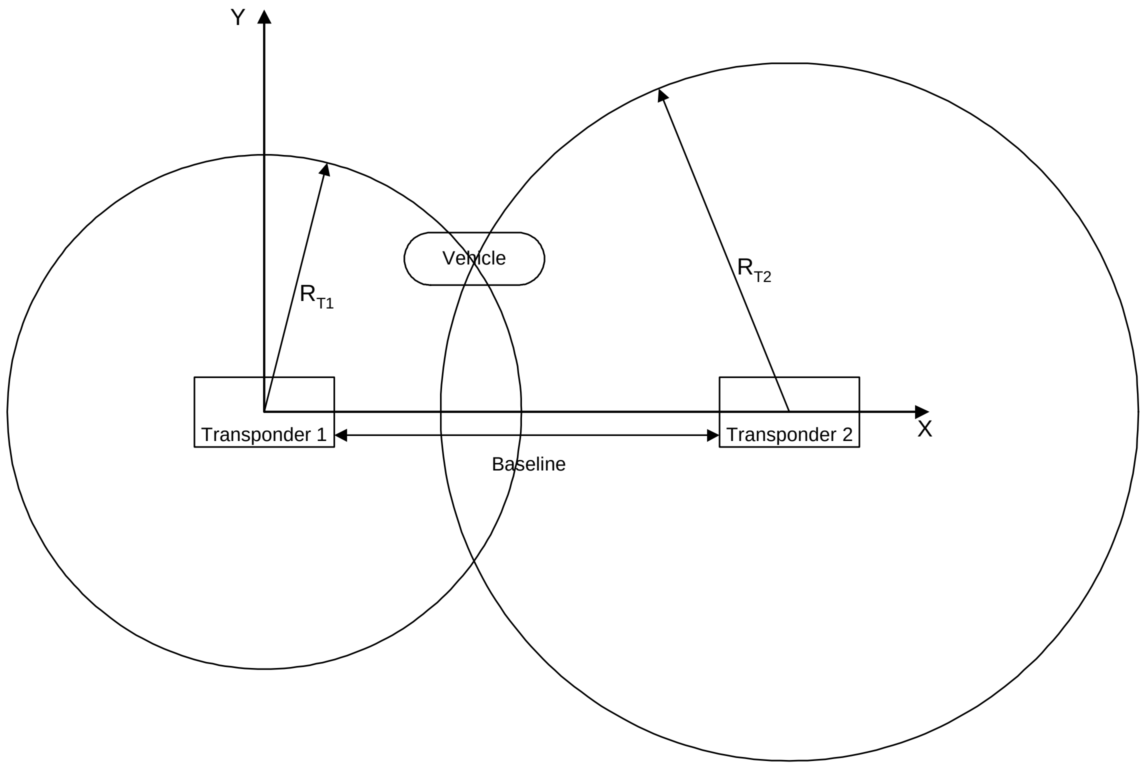


Figure 6. Acoustic long-baseline navigation systems use range readings to triangulate the vehicle's position. Vehicles typically operate on only one side of the x-axis to eliminate the ambiguity resulting from the range rings intersecting in two locations.

There are a variety of possible errors associated with the use of an acoustic LBL system. For the purpose of this work, the errors are typically placed into one of two categories. The first category consists of errors that equally affect all vehicles using the LBL system and therefore have minimal impact on the relative-navigation errors. One such error might be the incorrect placement of the LBL transducers, which causes a constant bias for all vehicles using the LBL. The second category includes errors that only affect individual vehicles in the group, such as errors resulting from internal clock errors, channel noise, etc., which impact the relative-navigation errors between the vehicles in the group.

d. Global Positioning System (GPS)

GPS is a space-based navigation system that uses triangulation to calculate the position of receivers located on or near the surface of the earth. The distance of the receiver from each satellite is determined using a time of flight scheme. The satellites periodically transmit messages containing the position of the satellite (ephemeris), timing, and status information. The distance (pseudorange) to each satellite is determined by measuring the amount of time it takes these messages to reach the receiver. By using the position of the satellites and the distance of the receiver from the satellites, the receiver can triangulate its position as shown in Figure 7.

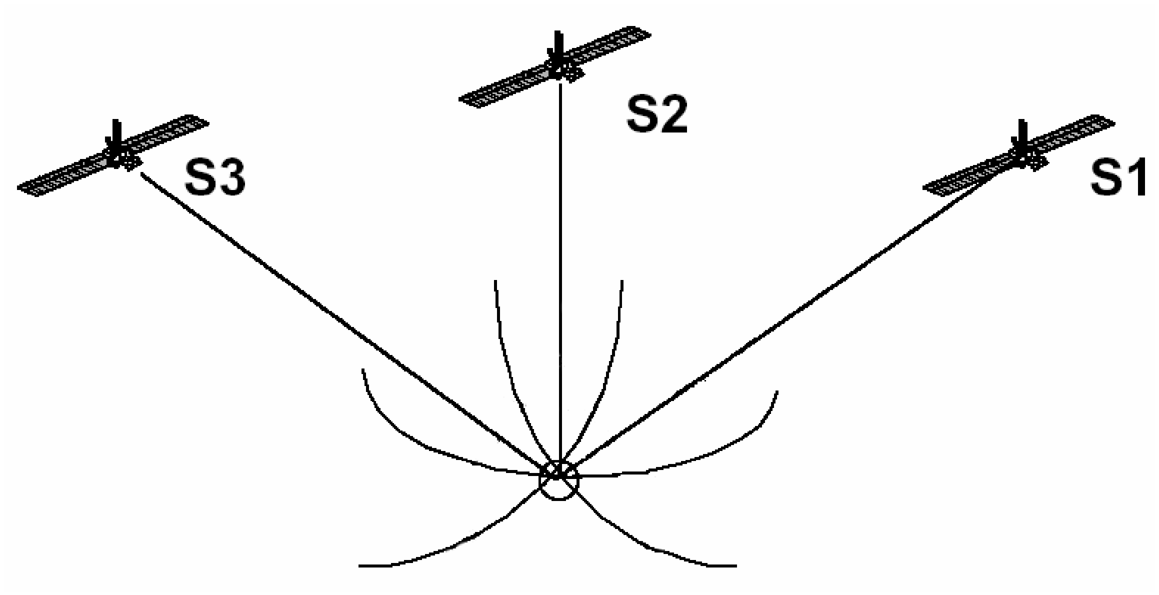


Figure 7. GPS uses pseudoranges and ephemeris information to triangulate the positions of receivers located on or near the surface of the earth (from University of Pennsylvania NROTC 2003).

Data is transmitted from the GPS satellites to the user's receiver using two L-Band carrier frequencies 1575.42 MHz (L1) and 1227.60 MHz (L2), which are not able to penetrate the water. As a result, AUVs equipped with GPS must periodically perform what is known as a GPS pop-up, as shown in Figure 8. During a pop-up the vehicle must surface, obtain at least one GPS fix, and then re-submerge to continue the mission.

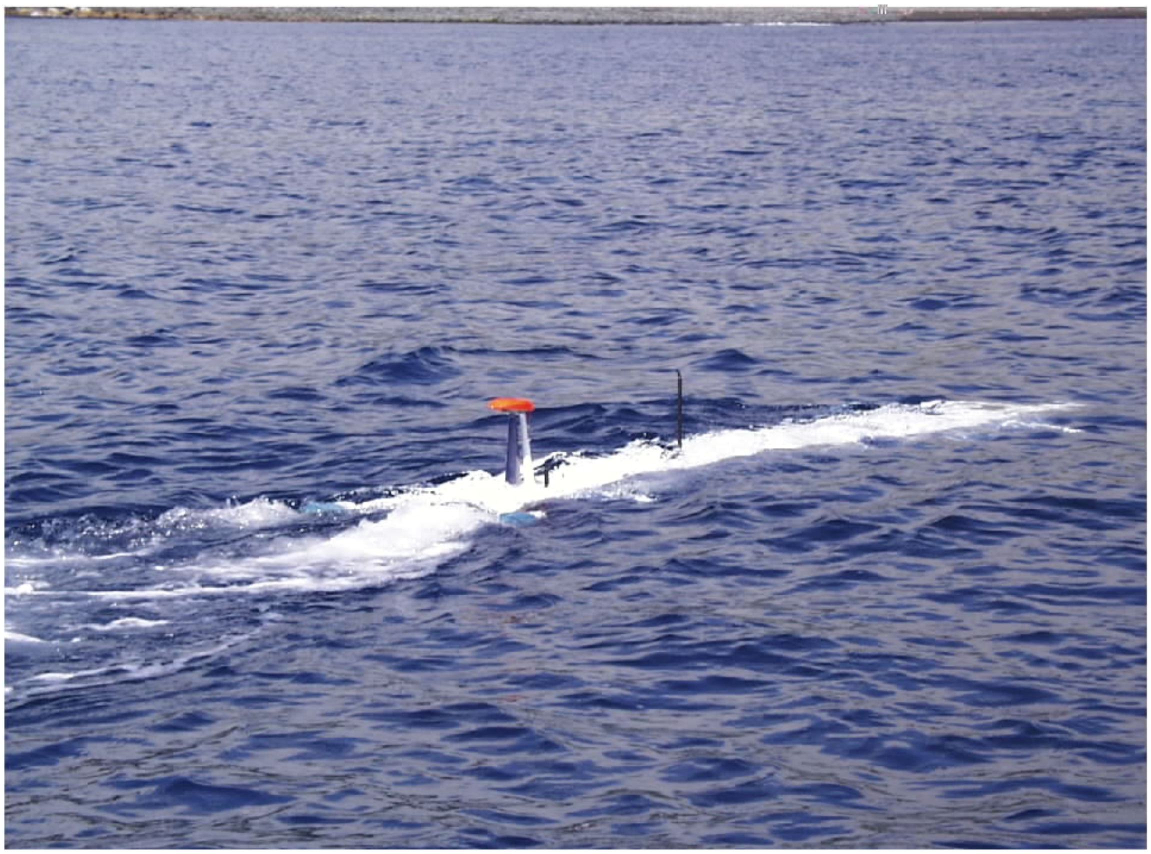


Figure 8. NPS' Acoustic Radio Interactive Exploratory Server (ARIES) AUV performing a GPS pop-up during Azores operations in August 2001 (from Nguyen 2002).

C. ACOUSTIC COMMUNICATIONS (ACOMMS)

1. Overview

Acoustic or telesonar modems provide bidirectional half-duplex wireless underwater communications, which allows multiple AUVs to share data without surfacing or being physically connected through a tether.

Acoustic modems typically receive data from the AUV's computer over a wired serial link, such as RS-232. The modem then frames the data and encodes frame, adding information for error detection and correction. Next the signal is modulated onto a carrier frequency and transmitted using a transducer, which converts the electrical signals into acoustic pressure waves. The pressure waves are recovered by the transducer connected to the receiving modem, which then demodulates the signal, decodes the data, corrects any errors, and transfers the received digital data to the receiving vehicle's computer.

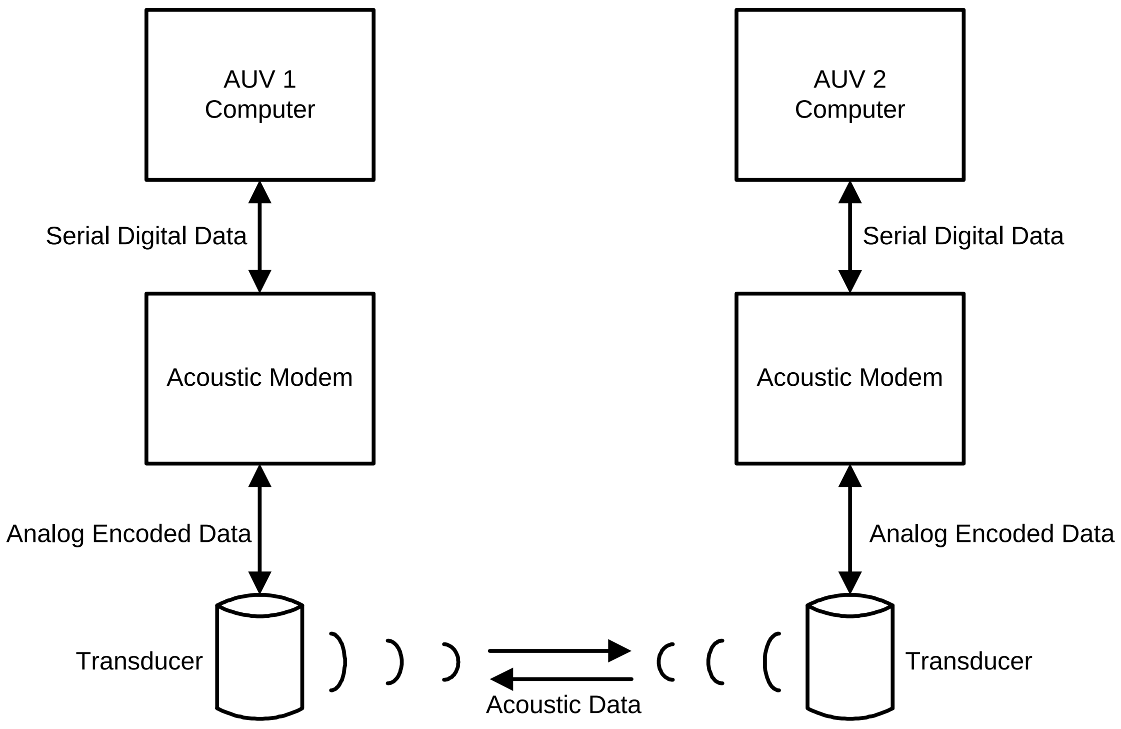


Figure 9. The basic operation and data flow for typical acoustic modems.

2. Ranging Function

Acoustic modems typically include a ranging feature that can determine the distance between two modems. This is an active system that requires the modem requesting the range to send a request to the modem in question, which in turn transmits a reply. The modem requesting the range measures the elapsed time between when the range request was sent and when the reply was received. The timing information is then adjusted to reflect the one-way time of flight, which is used in conjunction with the speed of sound in water to calculate the distance between the two transducers.

There are latencies associated with obtaining the range reading. For example, the Benthos 885 modems require several seconds to complete the ranging process even at short distances. Furthermore, user data cannot be transmitted while ranging requests are being processed because the half-duplex communications channel is utilized for the ranging operation.

Repeated performance of the ranging function permits the approximate computation of range rate. Some systems are further able to detect Doppler shift due to relative motion and thus estimate range rate directly.

3. Limitations

There are several limitations associated with the use of acoustic modems for underwater communications. Acoustic modems provide very limited bandwidth; for some modems the baud rate is 80 bps with best case conditions. Typically the communications reliability increases when the baud rate is decreased, so it may be necessary to reduce the baud rate based on the distance between the modems, background noise, depth, the position of the transducers relative to other underwater structures, or other similar factors. Moreover, acoustic modems are much more likely to experience drop-outs, dead zones, and corrupted message compared to today's RF modems.

The baud rate for an acoustic modem is usually selected based on environmental conditions and the distance between the transmitting and receiving modems. (Marr 2003) presents the results of "real-world" acoustic modem tests completed in Monterey Bay California. For all modem configurations tested it was found that the distance between the sending and receiving modems must be minimized to in order to achieve the maximum baud rate. Figures 10, 11, and 12 summarize the results of the modem experiments completed using Benthos 89x modems for various configurations and water depths.

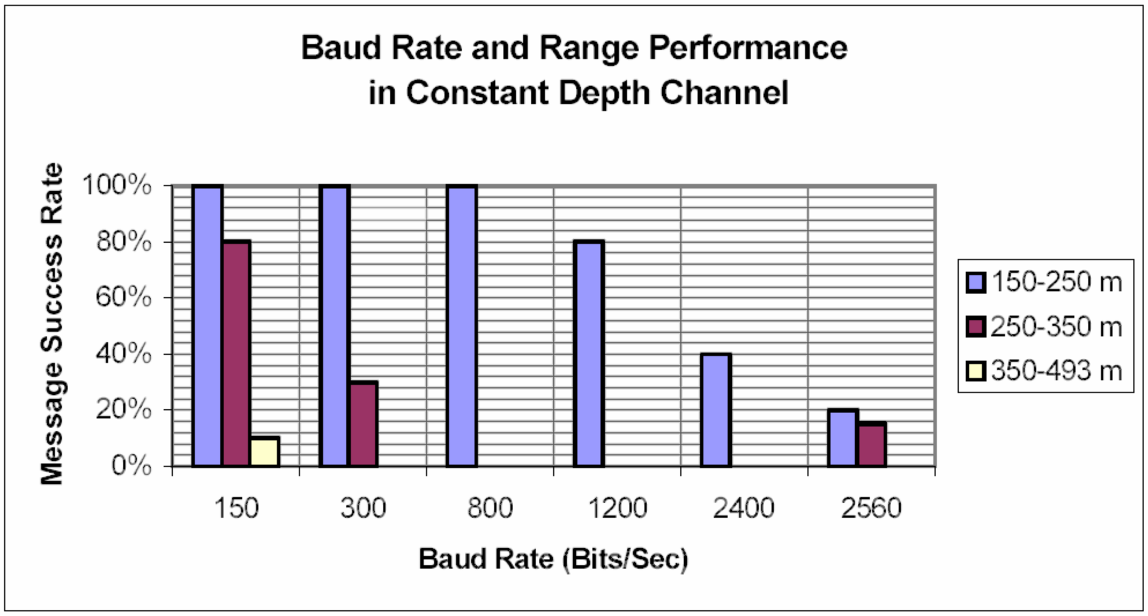


Figure 10. Benthos 89x acoustic modem performance in 15m of water with the transducers placed at a depth of 8m. (from Marr 2003)

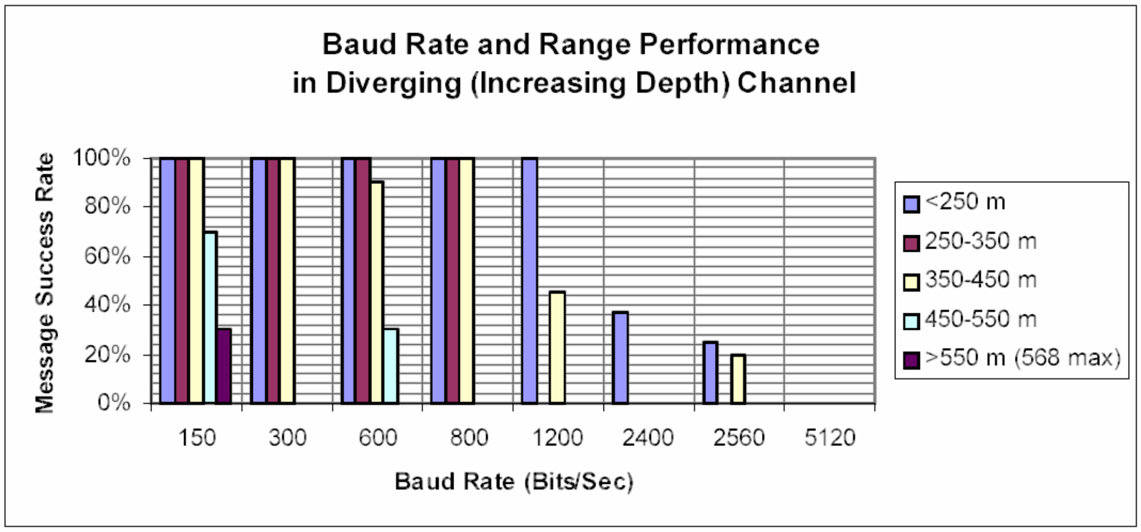


Figure 11. Benthos 89x acoustic modems performance with a diverging channel where the data is sent from approximately 15m of water to a modem in approximately 30m of water with both modem transducers placed at a depth of 8m. (from Marr 2003)

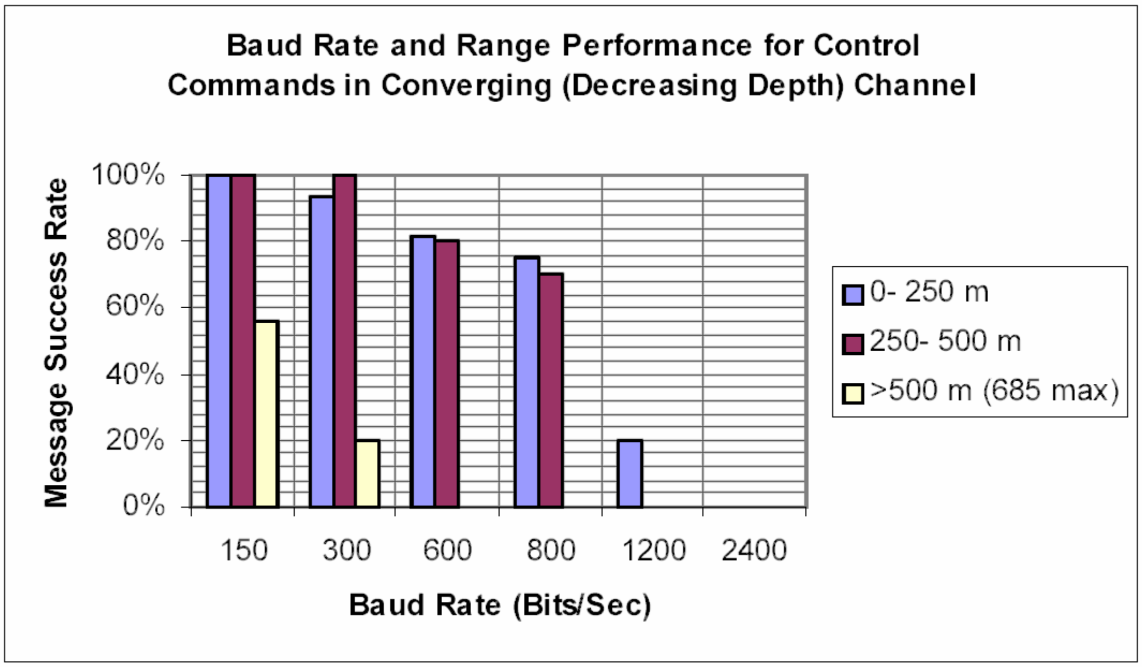


Figure 12. Benthos 89x acoustic modems performance with a converging channel where the data is sent from approximately 30m of water to a modem in approximately 15m of water with both modem transducers placed at a depth of 8m. (from Marr 2003)

Research has been completed to design methods of reducing the number of corrupted messages received using acoustic modems. For example, (Reimers 1995) presents a method of using forward error detection and error correction (FEC) coding to correct bit-errors in received messages. Despite the increased bandwidth required to transmit the FEC coding, (Reimers 1995) proposes a viable solution for the transmission of critical data or for coping with excessive bit-errors that are the result of environmental noise, fading signal strength, or other similar conditions.

D. COORDINATE SYSTEM DEFINITIONS

1. Overview

There are two coordinate systems used when discussing AUV navigation, the global reference frame and the vehicle local reference frame. This section discusses these frames and their relationship to each other.

2. Global Reference Frame

The global reference frame, also known as the global navigation or the north / east / down (NED) reference frames, is the coordinate system common to all underwater vehicles in the group. The global reference frame is defined with some origin O located at the surface of the water and axes aligned in the north, east and down (NED) directions as shown in Figure 13.

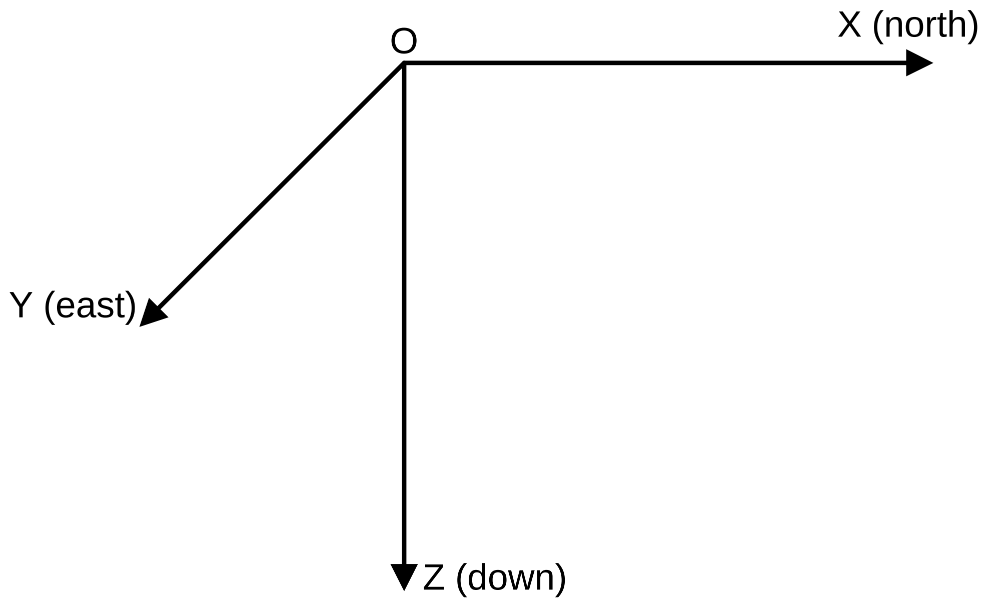


Figure 13. The global / NED reference frame has some origin O located at the surface of the water and axes aligned in the north, east, and down directions.

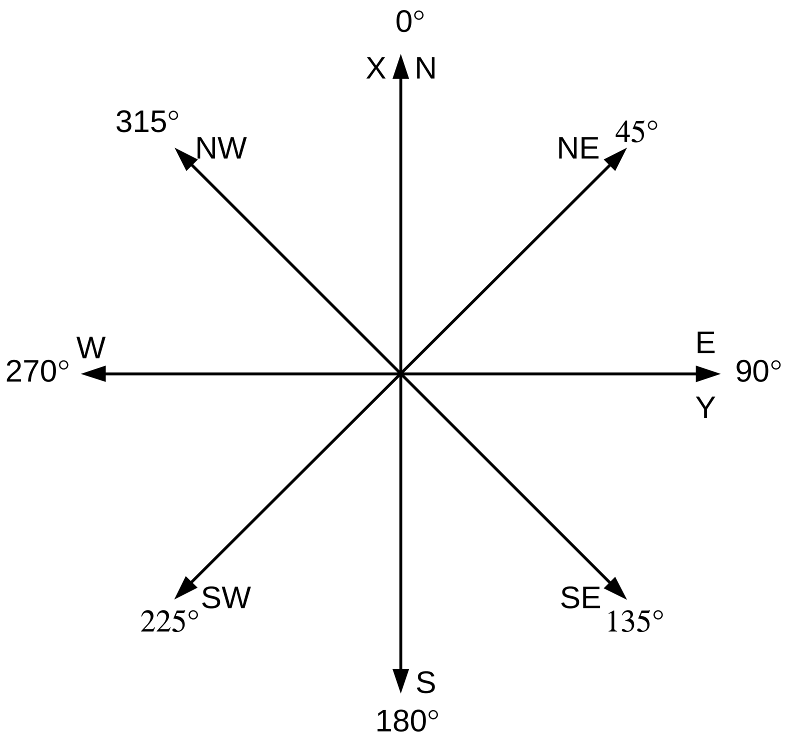


Figure 14. Angles measured in the horizontal plane of the global / NED reference frame, such as heading and course, are referenced to the north axis, such that 0-degrees represents north, 90-degrees is east, 180-degrees is south, and 270-degrees is west.

3. Body Fixed Frame of Reference

The body fixed frame of reference, also referred to as a vehicle's local reference frame, is unique to each vehicle in the group. The reference frame has an origin O' located on the centerline of the vehicle halfway between the bow and stern, which typically does not quite coincide with the vehicle's center of mass. The x-axis is along the center line pointing forward out of the bow, the positive y-axis starts at the origin and points out from the vehicle's starboard side, and the z-axis points down. It is important to note that the body-fixed reference frame moves and rotates to match changes in the vehicle's position and orientation.

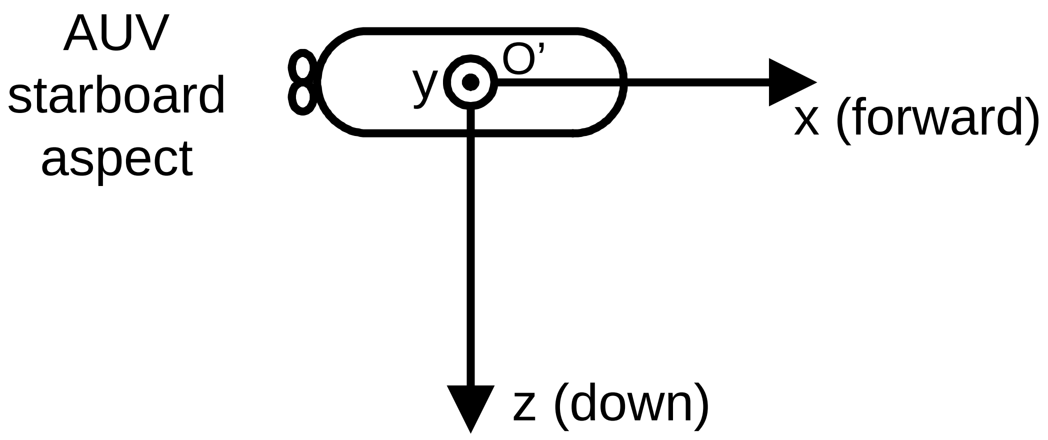


Figure 15. Starboard view of the body fixed reference frame with origin O’ and the y-axis coming out of the page.

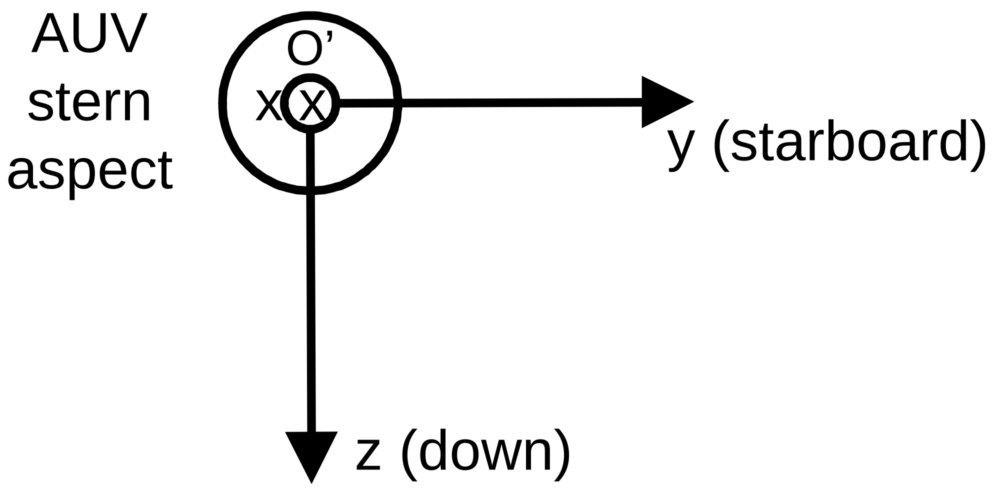


Figure 16. Stern view of the body fixed reference frame with origin O’ and the x-axis going into the page.

E. RELATIVE NAVIGATION ERROR CORRECTION

As discussed earlier, accumulated DR errors are often significant and therefore can quickly degrade the accuracy of a vehicle’s estimated position, in particular when an LBL or GPS is not used to periodically correct these errors. In context of cooperative behaviors, accumulated DR errors can have an even more profound impact. For example, assume two vehicles are performing follow the leader and the LV transmits its estimated position to the FV for the purpose of planning its course. The FV will then calculate its required course based on erroneous LV positional information that does not accurately reflect the LV's true position. The problem is further compounded because the FV will then use its own erroneous position estimate while attempting to follow the course. The errors for the LV and FV then become additive in the context of the follow-the-leader cooperative behavior, and the loss of acoustic contact or threat of collision can be the unexpected results.

There are several proposed solutions for correcting accumulated dead reckoning (DR) errors using range or range rate information. For example, (Alleyne 2000) proposes a solution that uses a Kalman filter and frequent range readings taken while the FV performs a zig-zag pattern to estimate the position of the FV relative to the LV. Solutions are proposed in (Dai 1997a) and (Dai 1997b) that use range-rate information to locate and track moving objects, which can be modified for the purpose of correcting the relative navigation errors between two vehicles.

One long-term goal of cooperative behaviors is to have large groups of vehicles simultaneously working together in an area to complete a mission. As a result there may be numerous vehicles sharing the limited bandwidth provided by acoustic modems. Furthermore, many commercially available modems use a portion of the bandwidth to request and receive ranging information. For these reasons it is essential that the vehicles minimize their communications and ranging requests, so as not to disrupt the cooperative behaviors being performed by other vehicles in the group. Additionally, the minimization of vehicle-to-vehicle communications further reduces the likelihood of the vehicles being detected by opposing forces. Many of the previously proposed solutions do not meet this requirement or have requirements that cannot realistically be met in the context of cooperative behaviors. For this reason, a solution for correcting the relative navigation errors between vehicles that uses only a small portion of the communications bandwidth and does not impose specific performance requirements on the vehicles is proposed in Chapter IV.

F. AUV WORKBENCH SIMULATOR

1. Overview

This section provides background information for the AUV Workbench, which is physics-based 3D virtual world AUV simulator and mission planning tool, which was enhanced as part of this thesis to support the development and evaluation of cooperative behaviors.

2. Supporting Technologies

a. Overview

The AUV Workbench uses several open-source software libraries and well established standards, which are discussed in this section.

b. Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a restricted form of the Standard Generalized Markup Language (SGML). XML provides a means of describing, validating, and structuring any type of data for storage or transmission over the internet. XML is not a language in the traditional sense, but instead provides a standard for creating languages to describe data for any application. The XML standard uses tags to mark and identify data, such as shown in Figure 17. Refer to (Hunter 2000), (Deitel 2001), or similar references for additional information regarding the XML standard.

```
<name>
        <first>John</first>
        <last>Smith</last>
</name>
```

Figure 17. XML example demonstrating the use of tags to store name information.

c. Extensible Style Sheet Language Transformations (XSLT)

Extensible style sheet transformations (XSLT) provide a means of transforming an XML document to another text format (XML or otherwise). XSLT uses XML-based style sheets, also known as templates, to interpret data contained in the input XML file and to define the output file format. Refer to (Hunter 2000), (Deitel 2001), and (Burke 2001) for additional information regarding XSLT.

d. Simple API for XML (SAX)

The simple API for XML (SAX) provides a method of reading and parsing XML files for use in application programs. SAX parsers are event driven, so when the parser encounters tags within the input XML document, it generates events that invoke user-written functions to process the data. For example, assume a SAX parser is used to read the XML file in Figure 17. Once the parser reaches the <name> tag, it calls a user-defined function to process the name information. Refer to (Hunter 2000) and (Deitel 2001) for additional information regarding SAX.

e. Virtual Reality Modeling Language (VRML)

The Virtual Reality Modeling Language (VRML) is a standard for defining 3D virtual worlds through the use of a structured text file, such as depicted in Figure 18. The text files are typically small and are ideal for transmission over the internet. VRML virtual worlds are rendered using specialized viewers that read the VRML text files and render the content defined in the file. There are several free VRML viewers available, such as Cortona and Cosmo Player, which are installed as internet browser plug-ins. There are also several open source VRML viewers available on the internet, such as Xj3D. Refer to (Ames 1997) for additional information regarding the VRML standard.

```
#VRML V2.0 utf8
NavigationInfo {
  type [ "EXAMINE" "ANY"  ]
}
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 1 1 1
    }
  }
  geometry Box {
    size 1 1 1
  }
}
```

Figure 18. VRML example defining a 1m x 1m x 1m white square.

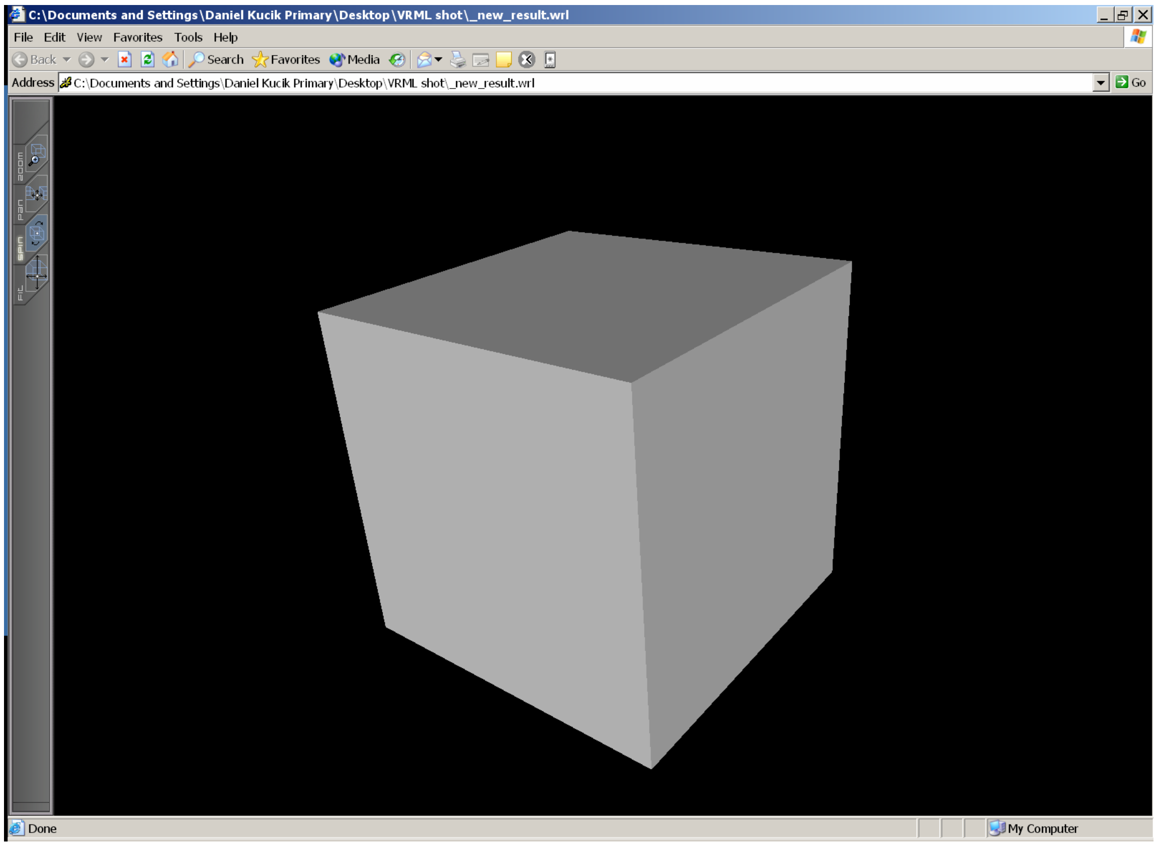

C:\Documents and Settings\Daniel Kucik Primary\Desktop\VRML shot_new_result.wrl
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
Go
Done
My Computer

Figure 19. Rendering of the 1m x 1m x 1m white square defined in Figure 18 using Internet Explorer and the Cortona VRML plug-in.

f. Extensible 3D (X3D) Graphics

Extensible 3D (X3D) Graphics is a language for defining 3D virtual worlds that is similar to VRML, except X3D conforms to the XML standard. There are numerous benefits to using X3D oppose to VRML, such as file validation and portability. X3D files are typically small and are optimized for transmission over the internet.

Several viewers are available that are capable of rendering 3D worlds using native X3D files, such as the Xj3D browser (Xj3D 2003). There are also several open-source tools available for developing X3D content and for translating X3D to/from VRML (Web3D 2003b). Refer to (Web3D 2003a) for more information regarding the X3D standard.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "http://www.web3d.org/specifications/x3d-3.0.dtd"
                "file:///www.web3d.org/TaskGroups/x3d/translation/x3d-3.0.dtd">
<X3D profile="Immersive">
  <head>
    <meta content="X3DBoxExample.x3d" name="filename"/>
    <meta content="Dan Kucik" name="author"/>
    <meta content="22 September 2003" name="created"/>
    <meta content="1m x 1m x 1m blue box" name="description"/>
  </head>
  <Scene>
    <NavigationInfo type=' "EXAMINE" "ANY" '/>
    <Shape>
      <Appearance>
        <Material diffuseColor="1 1 1"/>
      </Appearance>
      <Box size="1 1 1"/>
    </Shape>
  </Scene>
</X3D>
```

Figure 20. X3D equivalent of the 1m x 1m x 1m white box defined in Figure 18.

g. Extensible Java 3D (Xj3D) Browser

Extensible Java 3D (Xj3D) is a Web3D Consortium project focused on the development of open-source Java-based tools for VRML 97 and X3D. The goal of this effort is to develop the tools necessary to incorporate X3D and VRML content into any application (Xj3D 2003). Updates are distributed online and also on CD via a twice-yearly update of the X3D Software Development Kit (SDK). Refer to http://sdk.web3D.org for additional X3D SDK distribution information.

The Xj3D Browser is a standalone viewer capable of rendering scenes written in both VRML and X3D. Figure 21 shows the Xj3D rendering of the X3D file defined in Figure 20. For additional information regarding the Xj3D project refer to (Xj3D 2003).

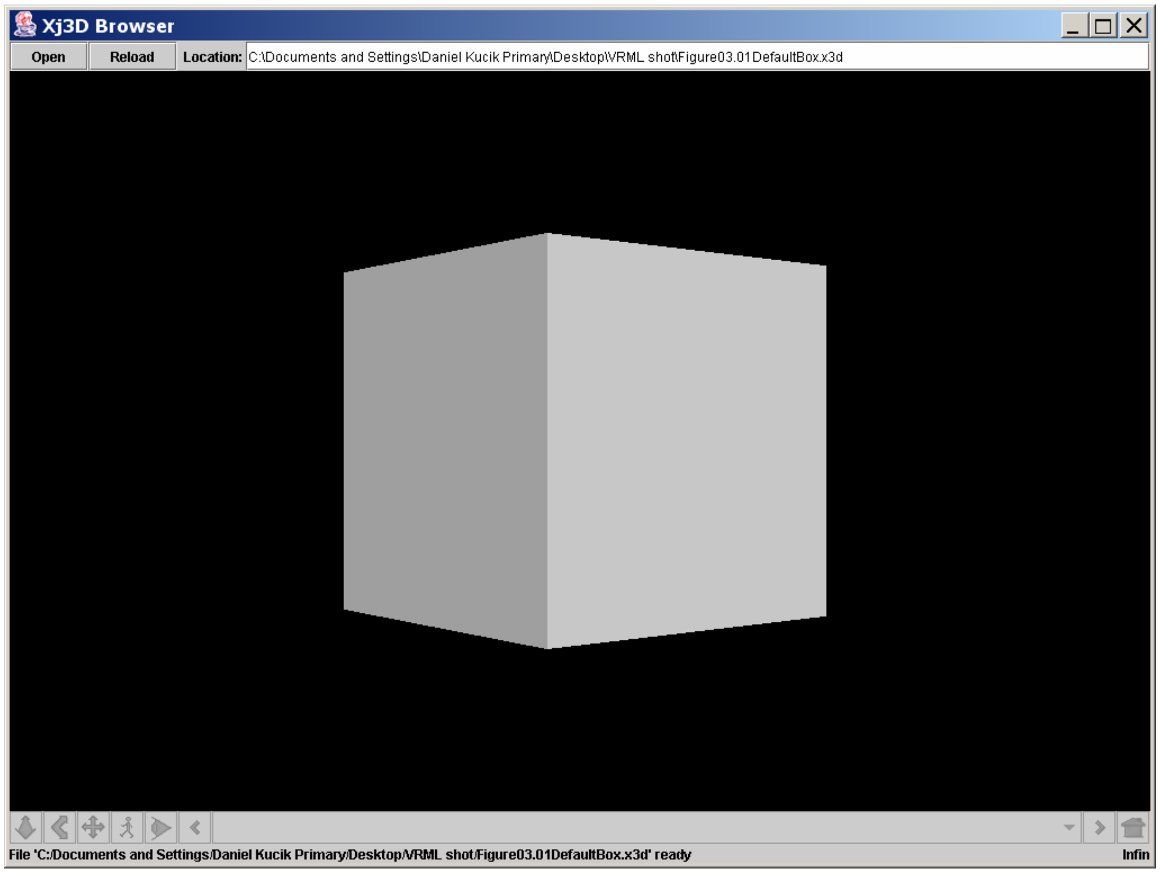
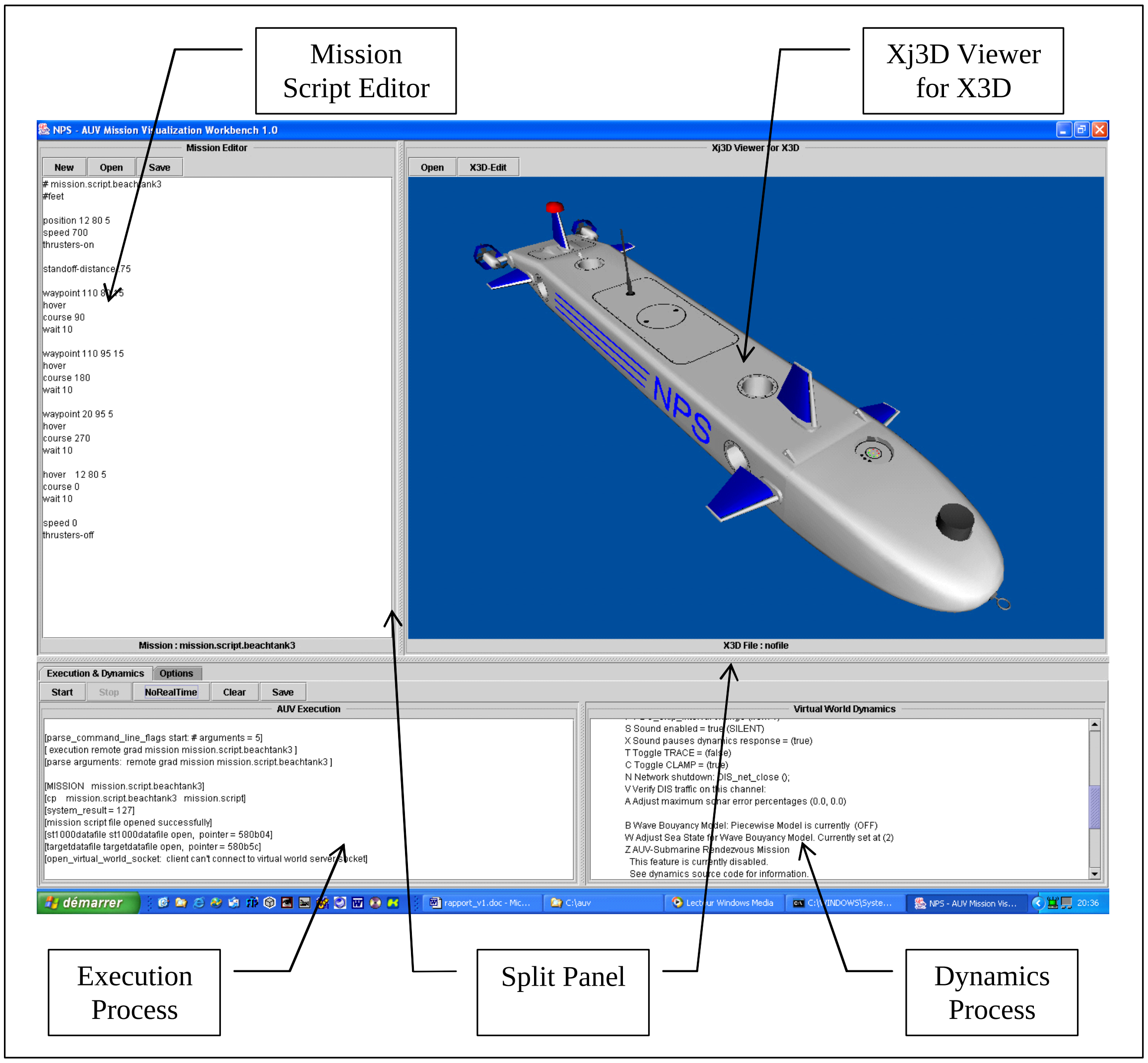

Xj3D Browser
Open
Reload
Location: C:\Documents and Settings\Daniel Kucik Primary\Desktop\VRML shot\Figure03.01DefaultBox.x3d
File 'C:/Documents and Settings/Daniel Kucik Primary/Desktop/VRML shot/Figure03.01DefaultBox.x3d' ready
Infin

Figure 21. Xj3D rendering of the X3D 1m x 1m x 1m white square defined in Figure 20.

3. AUV Workbench

The AUV Workbench is a physics-based 3D virtual-world simulator and mission-planning tool developed and built by NPS. It provides a means of writing, executing, and evaluating mission scripts before running the mission at sea with an AUV. There are many benefits to using the workbench. The most important benefit is the ability to test mission scripts before sea testing because it minimizes the time required at sea to troubleshoot "broken" scripts and it reduces the likelihood of vehicle damage or loss that may be the unexpected result incorrectly written scripts. The overall architecture of this virtual-world system is described in (Brutzman 1994)

The workbench is written in Java and uses open-source tools and libraries exclusively. During the execution of a mission, the workbench reads the mission script, simulates the vehicle's behavior using physics-based models, and renders the result in an Xj3D window, as depicted in Figure 22.



```
# mission.script.beachtank3
#feet

position 12 80 5
speed 700
thrusters-on

standoff-distance .75

waypoint 110 80 15
hover
course 90
wait 10

waypoint 110 95 15
hover
course 180
wait 10

waypoint 20 95 5
hover
course 270
wait 10

hover   12 80 5
course 0
wait 10

speed 0
thrusters-off
```

```
[parse_command_line_flags start: # arguments = 5]
[ execution remote grad mission mission.script.beachtank3 ]
[parse arguments:  remote grad mission mission.script.beachtank3 ]

[MISSION   mission.script.beachtank3]
[cp    mission.script.beachtank3   mission.script]
[system_result = 127]
[mission script file opened successfully]
[st1000datafile st1000datafile open,  pointer = 580b04]
[targetdatafile targetdatafile open,  pointer = 580b5c]
[open_virtual_world_socket:  client can't connect to virtual world server socket]
```

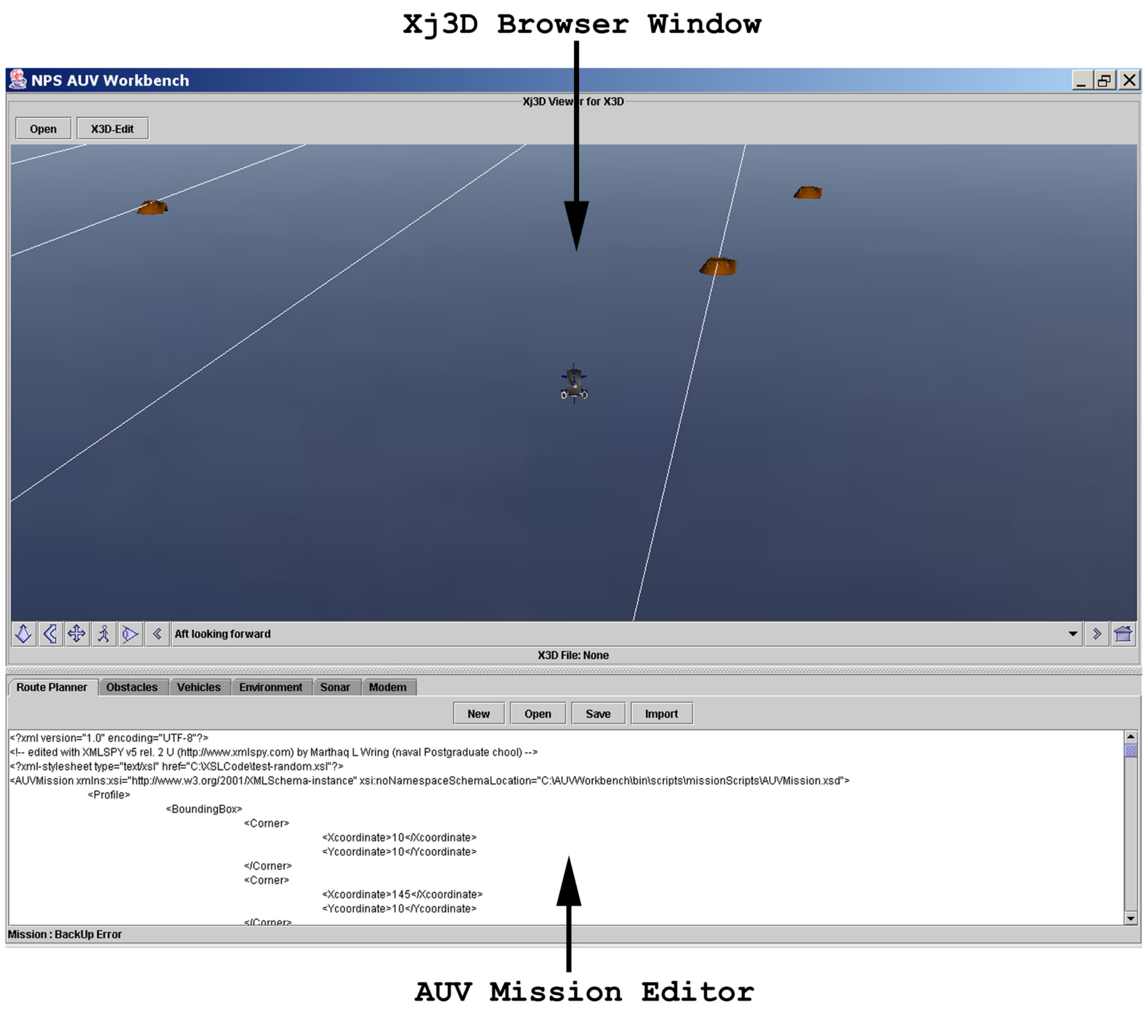
```
S Sound enabled = true (SILENT)
X Sound pauses dynamics response = (true)
T Toggle TRACE = (false)
C Toggle CLAMP = (true)
N Network shutdown: DIS_net_close ();
V Verify DIS traffic on this channel:
A Adjust maximum sonar error percentages (0.0, 0.0)

B Wave Bouyancy Model: Piecewise Model is currently  (OFF)
W Adjust Sea State for Wave Bouyancy Model. Currently set at (2)
Z AUV-Submarine Rendezvous Mission
  This feature is currently disabled.
  See dynamics source code for information.
```

Figure 22.   The graphical user interface (GUI) for the original AUV Workbench (from Grunesien 2002).

4. History and Contributors

The AUV Workbench is the result of the combined efforts of several past and present NPS students and faculty. The first version of the workbench, developed by Adrien Gruneisen and Yann Henriet (Grunesien 2002), and based on the dissertation research of Don Brutzman (Brutzman 1994), executes AUV missions while providing the user with a "close-up" view of the vehicle so the vehicle dynamics may be observed, as show in Figure 22. Doug Horner enhanced the workbench to include XML-based mission scripting, an obstacle avoidance algorithm, and modified the virtual world to provide a view of the work area to support mission planning, as depicted in Figure 23.

Refer to (Grunesien 2002) and (Brutzman 1994) for additional information regarding AUV simulation and the workbench.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by Marthaq L Wring (naval Postgraduate chool) -->
<?xml-stylesheet type="text/xsl" href="C:\XSLCode\test-random.xsl"?>
<AUVMission xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\AUVWorkbench\bin\scripts\missionScripts\AUVMission.xsd">
		<Profile>
				<BoundingBox>
						<Corner>
								<Xcoordinate>10</Xcoordinate>
								<Ycoordinate>10</Ycoordinate>
						</Corner>
						<Corner>
								<Xcoordinate>145</Xcoordinate>
								<Ycoordinate>10</Ycoordinate>
						</Corner>
```

Figure 23. Mission planning version of the AUV Workbench showing ARIES using its obstacle avoidance algorithm to maneuver around Manta mines.

G. SUMMARY

AUVs are unmanned underwater vehicles capable of performing a wide variety of missions, such as MCM and oceanographic surveys, with little or no operator intervention. Many AUVs are equipped with acoustic modems, which enable the vehicles to share near real-time information and to determine their ranges to other vehicles operating in the same area. Research efforts are underway to improve the efficiency and effectiveness missions completed using groups of AUVs through the use of cooperative behaviors, which enables the vehicles to coordinate their activities using information shared via acoustic modems. The AUV Workbench is an open-source physics-based virtual world simulator and mission planning tool for AUVs that has been modified for this thesis to support the development of cooperative behaviors.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TRIANGULATION THEORY: GEOMETRY AND EQUATIONS

A. INTRODUCTION

This chapter explains the theory and equations used by a number of operational LBL systems to triangulate positions using range values, which is an essential component of the proposed relative-navigation error-correction algorithm presented in the next chapter. The simplest method of triangulation is the two transponder approach, where the vehicle's range from two surveyed points is used to determine the vehicle's position. It is also possible to triangulate the position of a vehicle using three or more transponders; however this approach requires the deployment of additional transponders and the equations are more CPU intensive than the equations for the two transponder approach. When using either LBL configuration it is essential that the vehicle operates in the ideal operating area for that particular system to ensure the position solutions are observable.

B. TRIANGULATION USING TWO RANGE VALUES

1. Overview

This section presents the theory and equations for triangulating the position of a vehicle using two range values.

2. Assumptions

The method discussed in this section is capable of determining the two-dimensional (2D) position of a vehicle relative to one of the transponders and the fix is only accurate on one side of the line connecting the two transponders, also know as the baseline. Furthermore, to minimize the vertical component included in the ranging value that results from the vehicle and transponders being at different depths, it is assumed that vehicle depth is approximately equal to the depth of the transponders.

3. Triangulation Theory: Geometry and Equations

a. Overview

This section discusses the theory behind triangulation and presents a simplified set of equations for calculating positions using two range values.

b. System Configuration and the Triangulation Reference Frame

The LBL system requires that two transponders are placed at fixed positions that are typically surveyed using a GPS receiver. The positions of the transponders are then used to define a local LBL reference frame such as depicted in Figure 24, where the line connecting the two transponders, also known as the baseline, defines the x-axis and the y-axis is perpendicular to the baseline with the origin being one of the two transponders. It is important to note that the LBL reference frame is entirely dependent on the placement of the transponders, so this reference frame is typically not aligned with the global NED reference frame.

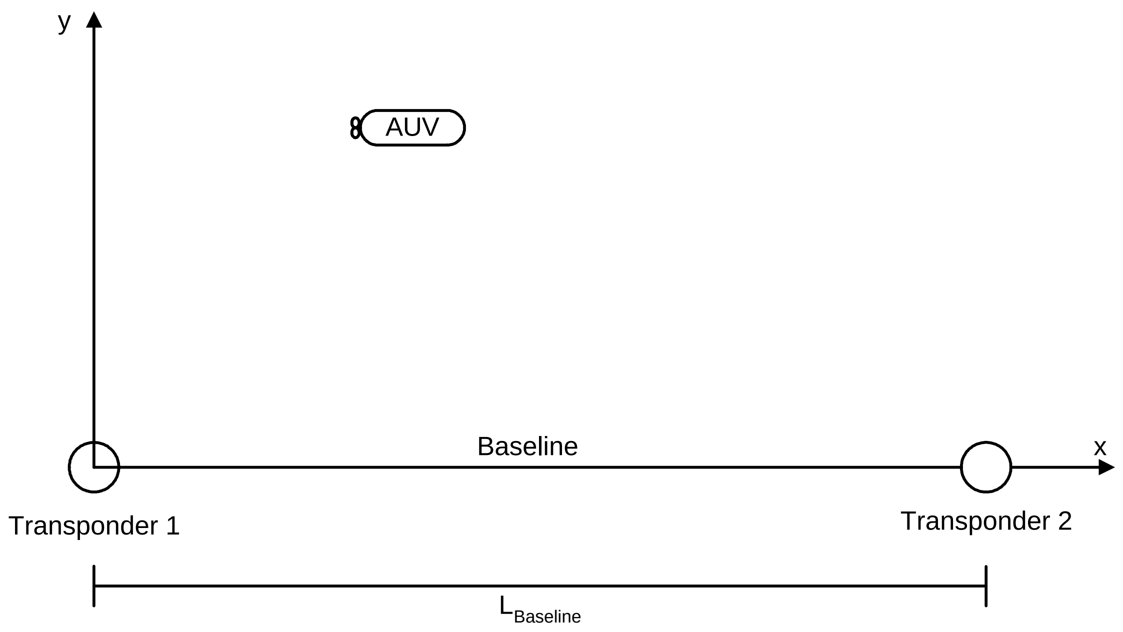


Figure 24. The triangulation reference frame where the origin is located at transponder 1, the x-axis is aligned with the baseline and runs through transponder 2, and the y-axis is perpendicular to the baseline and is positive in the direction of the work area.

LBL systems that use only two transponders require that the vehicles operate on only one side of the baseline, otherwise the ambiguity depicted in Figure 25 will occur. Some vehicles utilize their dead reckoning sensors in combination with the LBL system to determine when a baseline crossing occurs so they can adjust the LBL position fixes to compensate for the baseline crossing, and therefore these vehicles can operate on either side of the baseline.

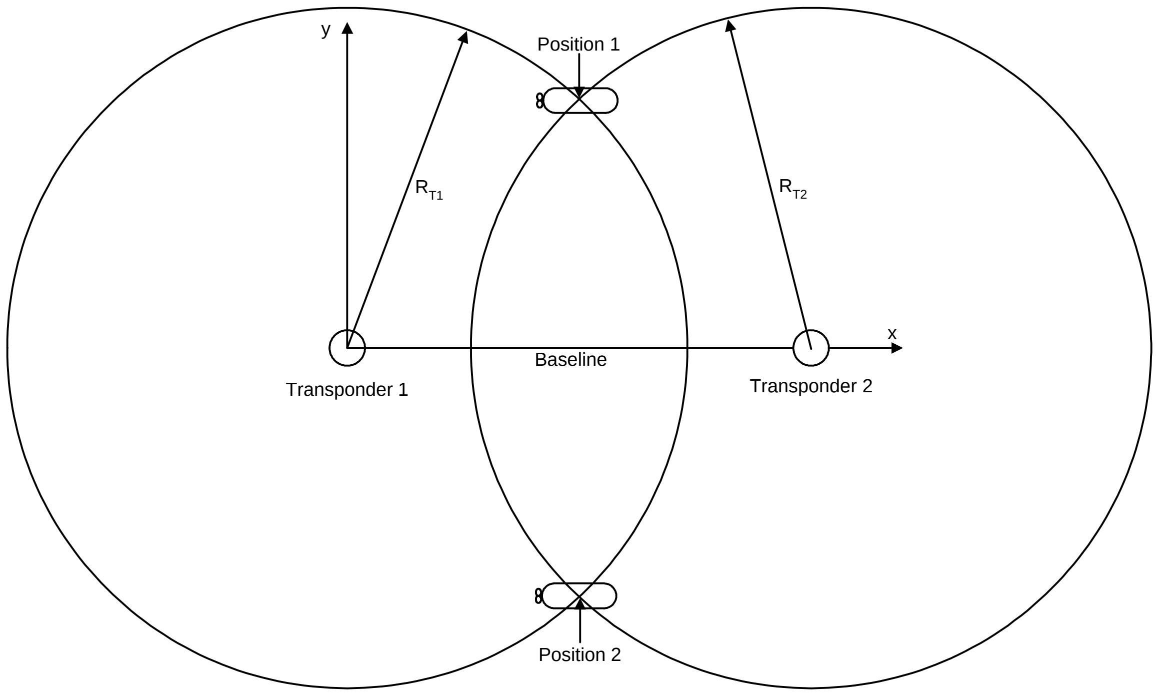


Figure 25. Baseline crossings result in ambiguities because the range rings centered on the transponders intersect in two places, so the LBL system is not able to determine if the vehicle is located at Position 1 or Position 2.

c. Triangulation Equations

Vehicles using the LBL system periodically receive range readings from the two transponders, which are represented as range rings centered on the transponders as shown in Figure 26. After eliminating the secondary intersection of the range rings located on the "wrong" side of the baseline, the ranging data can be simplified as shown in Figure 27. Figure 28 shows a further simplified version of the problem with the x and y components of the vehicle's position represented. Equation 2 and 3 are the Law of Cosines and a general trigonometric equation respectively, which are combined to produce Equation 4. Equation 4 uses the range readings from the two transponders and the baseline length to calculate the x-position of the vehicle. The y-position of the vehicle is calculated using equation 5, which is the Pythagorean Theorem.

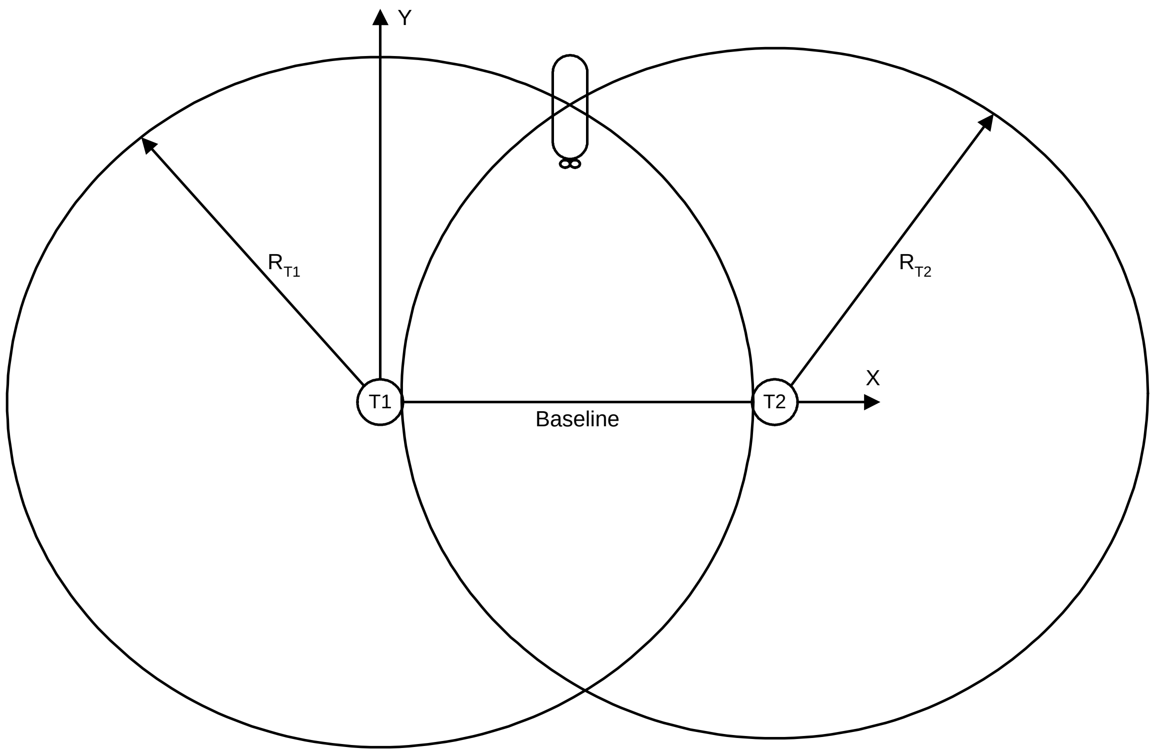


Figure 26. Range readings (R_{T1} and R_{T2}) obtained from the transponders represented as range rings centered on the transponders.

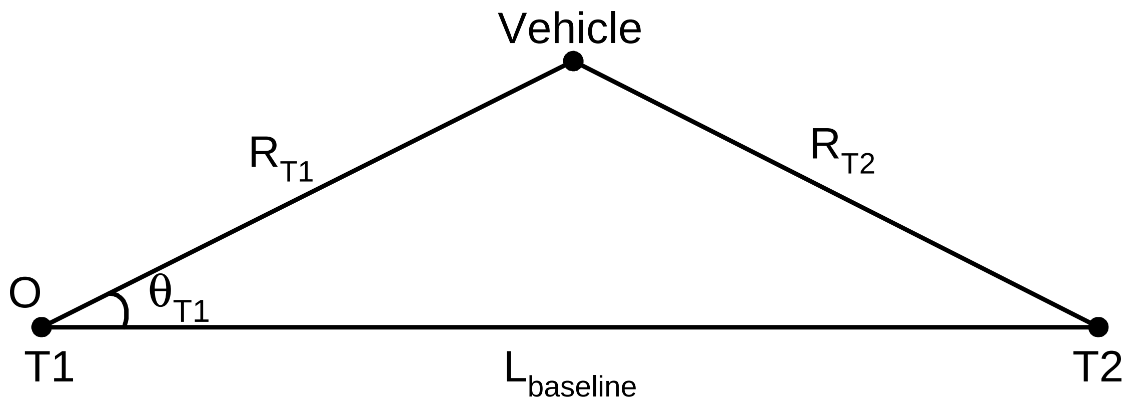


Figure 27. A simplified interpretation of the range readings after the exclusion of the secondary intersection of the range rings located on the “wrong” side of the baseline.

$$L_{baseline} = \sqrt{(T1_N - T2_N)^2 + (T1_E - T2_E)^2}$$

Equation 1: Calculation of the baseline length, which is the distance between transponders 1 and 2.

where,
$L_{baseline}$ = The distance between transponders 1 and 2.
$T1_N$, $T1_E$ = The north and east components of transponder 1's position in the NED reference frame.
$TP2_N$, $TP2_E$ = The north and east components of transponder 2's position in the NED reference frame.

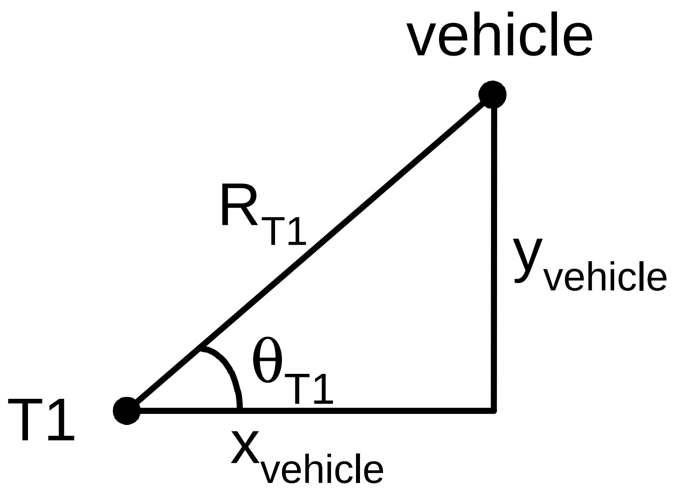


Figure 28. Simplified triangulation computation using the x and y values of the vehicle's position.

$$R_{T1}\cos q_{T1} = \frac{R_{T1}{}^2 + L_{baseline}{}^2 - R_{T2}{}^2}{2L_{baseline}}$$

Equation 2: The Law of Cosines equation that is combined with Equation 3 to calculate the vehicle's x-position in the triangulation reference frame.

where,
R_{T1} = The range between the vehicle and transponder 1.
R_{T2} = The range between the vehicle and transponder 2.
θ_{T1} = The angle between the triangulation frame's x-axis and the line connecting transponder 1 to the vehicle (as shown in Figure 27).
$L_{baseline}$ = The distance between transponders 1 and 2 (from Equation 1).

$$x_{vehicle} = R_{T1}\cos q_{T1}$$

Equation 3: Trigonometric relationship which is combined with Equation 2 to calculate the vehicle's x-position in the triangulation frame.

where,
$x_{vehicle}$ = the x-position of the vehicle in the triangulation reference frame.
R_{T1} = The range between the vehicle and transponder 1.
θ_{T1} = The angle between the triangulation frame's x-axis and the line connecting transponder 1 to the vehicle (as shown in Figure 28).

$$x_{vehicle} = \frac{R_{T1}^2 + L_{baseline}^2 - R_{T2}^2}{2L_{baseline}}$$

Equation 4: Calculation of the vehicle's x-position in the triangulation reference frame using the ranging data obtained from two transponders (McTrusty 2000).

where,
$x_{vehicle}$ = the x-position of the vehicle in the triangulation reference frame.
R_{T1} = The range between the vehicle and transponder 1.
R_{T2} = The range between the vehicle and transponder 2.
$L_{baseline}$ = The distance between transponders 1 and 2 (from Equation 1).

$$y_{vehicle} = \sqrt{R_{T1}^2 - x_{vehicle}^2}$$

Equation 5: Calculation of the vehicle's y-position in the triangulation reference frame using ranging information and the x-position calculated using Equation 4 (McTrusty 2000).

where,
$y_{vehicle}$ = the y-position of the vehicle in the triangulation reference frame.
$x_{vehicle}$ = the x-position of the vehicle in the triangulation reference frame (from Equation 4).
R_{T1} = The range between the vehicle and transponder 1.

4. Conversion to the Global Reference Frame

a. Overview

This section defines the equations necessary for converting LBL position fixes from the triangulation reference frame to the NED reference frame.

b. Triangulation Frame with Transponder 1 as the Origin

This section presents the equations for converting position fixes from the triangulation frame with transponder 1, as shown in Figure 26, as the origin to the NED reference frame. The orientation of the triangulation frame relative to the NED reference frame is depicted in Figure 29. The conversion requires the triangulation frame to be rotated and translated to match the NED frame. First it is necessary to determine the angle between the triangulation frame's x-axis and the NED east-axis using Equation 6. The triangulation frame is then rotated using Equation 7 and translated using Equation 8.

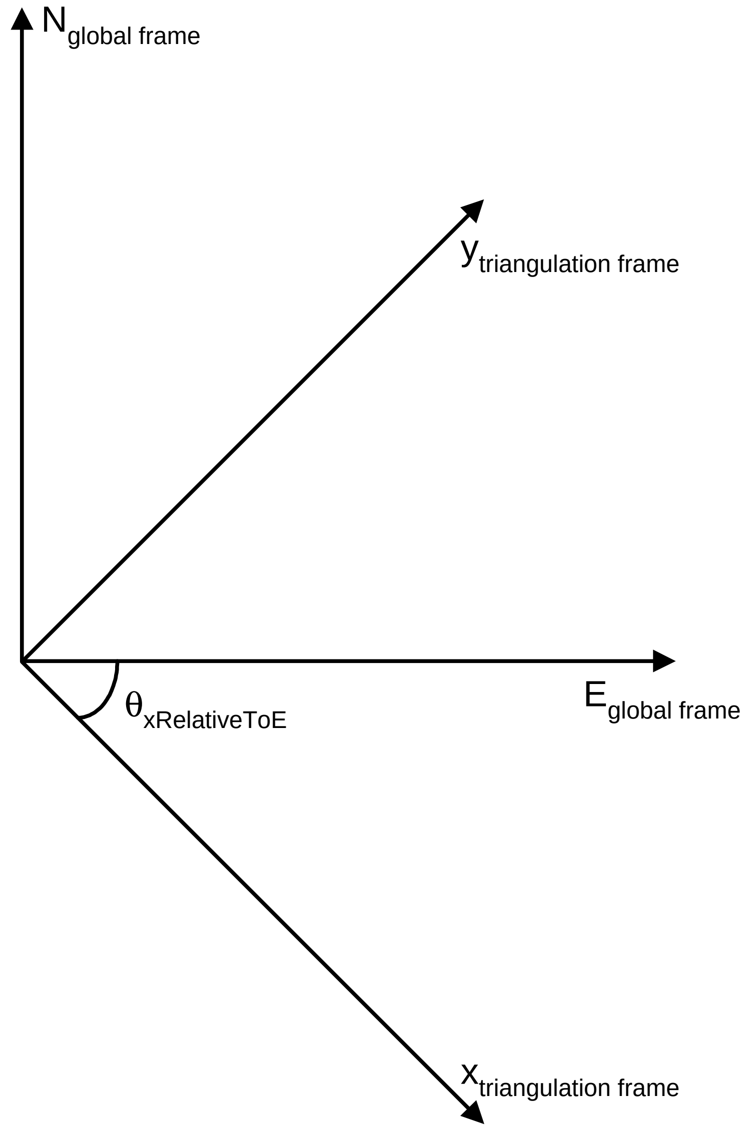


Figure 29. Triangulation frame's orientation relative to the NED frame with transponder 1 as the origin of the triangulation frame.

$$q_{x\mathrm{Re}lToE} = \mathrm{atan2}((T2_N - T1_N),(T2_E - T1_E))$$

Equation 6: Calculation of the angle between the correction frame's x-axis and the NED east axis.

where,
$\theta_{xRelToE}$ = The angle between the triangulation frame's x-axis and the NED frame's east axis.
$T1_N$, $T1_E$ = The north and east components of transponder 1's position in the NED frame.
$T2_N$, $T2_E$ = The north and east components of transponder 2's position in the NED frame.

$$\begin{bmatrix} vehicle_{E/T1} & vehicle_{N/T1} & 1 \end{bmatrix} = \begin{bmatrix} x_{vehicle} & y_{vehicle} & 1 \end{bmatrix} * \begin{bmatrix} \cos q_{x\,\mathrm{Re}\,lToE} & \sin q_{x\,\mathrm{Re}\,lToE} & 0 \\ -\sin q_{x\,\mathrm{Re}\,lToE} & \cos q_{x\,\mathrm{Re}\,lToE} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 7: Rotation of triangulation reference frame to match the NED reference frame (Healey 2003).

where,
$vehicle_{N/T1}$, $vehicle_{E/T1}$ = The north and east components of the vehicle's position with transponder 1 as the origin. Note that the east and north order indicated in the equation is intentional to account for the triangulation frame's x and y axes being aligned with the NED east and north axes respectively. This approach eliminates the need for a second rotation, therefore reducing the CPU time required to perform the conversion.
$x_{vehicle}$, $y_{vehicle}$ = The x and y components of the vehicle's position in the triangulation frame. (from Equations 4 and 5).
$\theta_{xRelToE}$ = The angle between the triangulation frame's x-axis and the NED frame's east axis (from Equation 6).

$$\begin{bmatrix} vehicle_N & vehicle_E \end{bmatrix} = \begin{bmatrix} vehicle_{N/T1} & vehicle_{E/T1} \end{bmatrix} + \begin{bmatrix} T1_N & T1_E \end{bmatrix}$$

Equation 8: Translation of the triangulation frame to match the NED reference frame's origin.

where,
$vehicle_N$, $vehicle_E$ = the vehicle's north and east position in the NED reference frame.
$vehicle_{N/T1}$, $vehicle_{E/T1}$ = The north and east components of the vehicle's position with transponder 1 as the origin (from Equation 7).
$T1_N$, $T1_E$ = The north and east components of transponder 1's position in the NED frame.

c. Triangulation Frame with Transponder 2 as the Origin

This section presents the equations for converting position fixes from the triangulation frame with transponder 2, as shown in Figure 26, as the origin to the NED reference frame. The orientation of the triangulation frame relative to the NED reference frame is depicted in Figure 30. The conversion requires the triangulation frame to be rotated and translated to match the NED frame. First it is necessary to determine the angle between the triangulation frame's x-axis and the NED north-axis using Equation 9. The triangulation frame is then rotated using Equation 10 and translated using Equation 8 presented in the previous section.

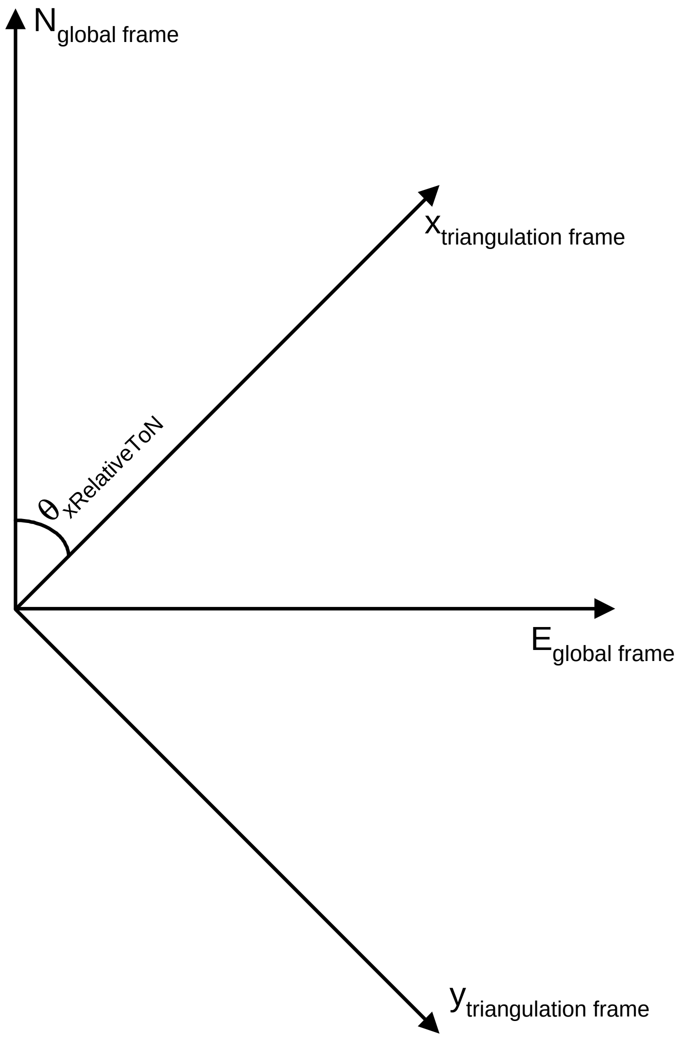


Figure 30. Triangulation frame's orientation relative to the NED frame with transponder 2 as the origin of the triangulation frame.

$$q_{x\,\mathrm{Re}\,lToN} = a\tan 2((T2_E - T1_E),(T2_N - T1_N))$$

Equation 9: Calculation of the angle between the correction frame's x-axis and the NED north axis.

where,
$\theta_{xRelToN}$ = The angle between the triangulation frame's x-axis and the NED frame's north axis.
$T1_N$, $T1_E$ = The north and east components of transponder 1's position in the NED frame.
$T2_N$, $T2_E$ = The north and east components of transponder 2's position in the NED frame.

$$[vehicle_{N/T1} \quad vehicle_{E/T1} \quad 1] = [x_{vehicle} \quad y_{vehicle} \quad 1] * \begin{bmatrix} \cos q_{x\mathrm{Re}lToE} & \sin q_{x\mathrm{Re}lToE} & 0 \\ -\sin q_{x\mathrm{Re}lToE} & \cos q_{x\mathrm{Re}lToE} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 10: Rotation of triangulation reference frame to match the NED reference frame (Healey 2003).

where,
$vehicle_{N/T1}$, $vehicle_{E/T1}$ = The north and east components of the vehicle's position with transponder 1 as the origin.
$x_{vehicle}$, $y_{vehicle}$ = The x and y components of the vehicle's position in the triangulation frame. (from Equations 4 and 5).
$\theta_{xRelToE}$ = The angle between the triangulation frame's x-axis and the NED frame's east axis (from Equation 9).

C. TRIANGULATION USING THREE OR MORE TRANSPONDERS

This section presents a second triangulation technique that uses three or more transponders. The equations used to calculate the vehicle's position for this approach are more CPU intensive than the algorithms presented for the two-transponder method. However, this approach has several benefits, such as allowing the vehicles to operate in any quadrant and it provides position fixes in the NED reference frame, whereas with the simplified two-transponder approach discussed in the previous section it is necessary to rotate and translate the triangulation reference frame to obtain position fixes in the NED frame.

This method of triangulation requires three or more range readings, as shown in Figure 31. A system of n equations is established based on Equation 11, where n is the number of transponder used by the system.

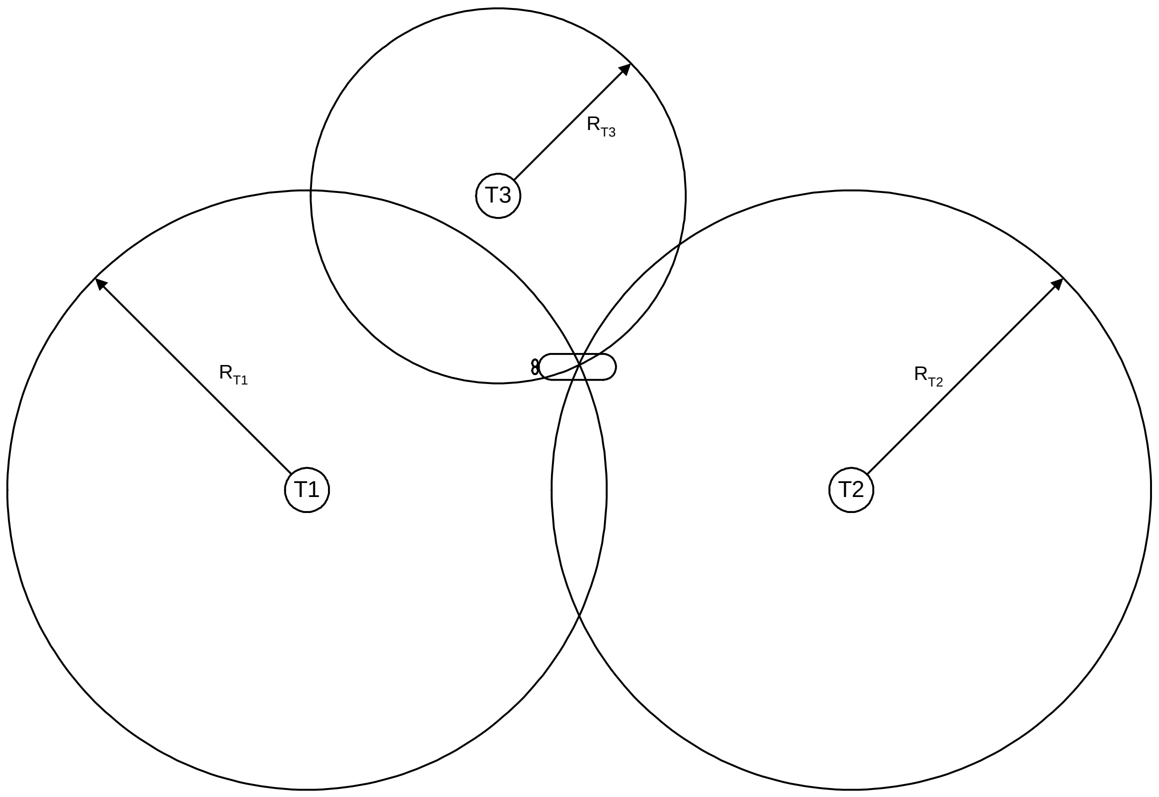


Figure 31. Position fixes calculated using triangulation techniques with three or more transponder can be represented as the intersection of the range rings centered on the transponders.

$$R_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}$$

Equation 11: Equation for triangulating positions using three or more range values.

where,
i = 1, 2, …, n, where n is the number of transponders.
R_i = the range reading obtained from transponder i.
x_i, y_i, z_i = the north, east, and down components of transponder i's position, which is known from the survey completed during the deployment of the transponders.
x, y = the calculated north and east components of the vehicle's position in the global NED reference frame.
z = the depth of the vehicle at the time of the position fix taken from the vehicle's depth sensor.

D. OPTIMAL TRIANGULATION GEOMETRY AND OBSERVABILITY

Triangulation-based systems, such as LBLs and GPS, require the appropriate transponder/vehicle geometry to obtain the maximum accuracy and to ensure observability. To meet this requirement, it is necessary that there is significant separation between the transponders and the vehicle, and the transponders and vehicle must not be collinear (Matos 1999), (Larsen 2000), and (Deffenbaugh 1996). Issues also arise when the vehicle is located far from the transponders because the line-of-sight (LOS) vectors become collinear, which diminishes the observability of the system (Larsen 2000).

In (Matos 1999) the ideal operating area for vehicles using a two transponder LBL system is defined as a rectangle with a minimum distance to the baseline of 1/4 of the baseline length and a maximum distance equal to the baseline length. The width of the optimal operating area is equal to the baseline length as shown in Figure 32. The vehicle can operate in an area slightly wider than the baseline length while still maintaining observability, but the accuracy of the triangulated position will degrade as the vehicle moves further away from the optimal operating area.

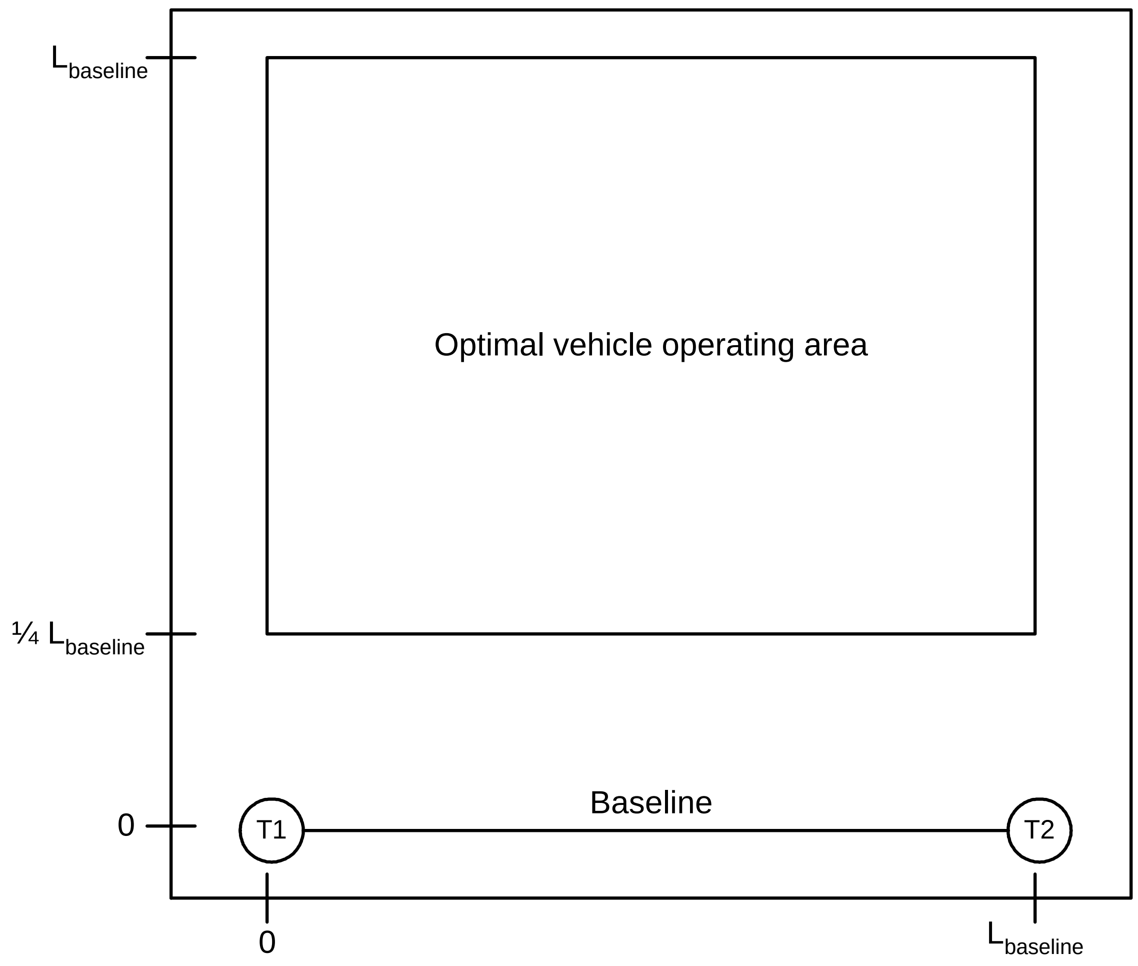


Figure 32. Optimal vehicle operating area for a two transponder LBL system, which ensures the triangulation solution is observable (Matos 1999).

E. SUMMARY

Triangulation is one of the most common methods for determining position, for example it is used by LBL systems, GPS, and other radio navigation aids, such as the LORAN system. Position fixes may be triangulated using only two range values, which minimizes the logistics of the system because only two transponders need to be deployed, and the required calculations do not require significant CPU time. Triangulation of position can also be completed using three or more range values, but this approach requires the deployment of additional transponders and the calculations are more CPU intensive than the two transponder approach. To ensure observability and to reduce the effects of geometric dilution of precision (GDOP), it is necessary to configure the LBL system such that the transponders baseline and vehicles are not collinear.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RELATIVE NAVIGATION ERROR CORRECTION

A. INTRODUCTION

This chapter proposes a cooperative behavior for correcting the relative-navigation errors between two AUVs that do not share a common means of automatically correcting their accumulated DR errors. Relative navigation errors are the result of accumulated DR errors for each vehicle in the group, and as discussed in Chapter II these errors may have negative effects on the performance of cooperative behaviors. There are several solutions for correcting the relative navigation errors between vehicles, however the approach proposed in this chapter is better suited for supporting cooperative behaviors performed with AUVs because it is not time intensive and does not require significant communications bandwidth. The proposed solution uses ranging and position data collected by the FV and the triangulation techniques described in Chapter III to correct the relative navigation errors between two vehicles.

B. FOLLOW-THE-LEADER TACTICS

1. Overview

The follow-the-leader behavior allows one vehicle (the following vehicle or FV) to autonomously follow a second vehicle (the lead vehicle or LV). There are two types of follow-the-leader tactics discussed here: track/path following and “relaxed” following.

2. Track/Path Following

During track/path following operations the FV attempts to duplicate the precise path taken by the leader such as shown in Figure 33. This type of vehicle following might be useful if a group of AUVs are transiting to a work area and only one vehicle has an obstacle avoidance sonar installed. In such a situation it is advisable to have all AUVs in the group follow the leader’s path so that even those vehicles without an obstacle avoidance sensor can avoid the obstacles detected by the leader. To successfully complete the track following behavior, it is necessary that the relative positioning error between the vehicles is minimal, making it better suited for vehicles that share a common long baseline navigation system, vehicles that can make frequent GPS pop-ups, or vehicles with highly accurate IMUs and DVLs installed.

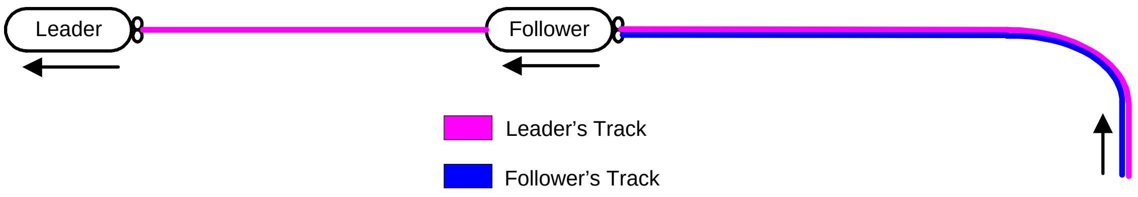


Figure 33. Track/path following operations where the following vehicle matches the precise track taken by the leader.

3. Relaxed Following

The constraints for relaxed following are less strict than those for track/path following. During relaxed following operations, the FV stays within a given minimum/maximum range bracket from the LV while loosely following the LV's track. This type of following is useful for in-transit high-speed data transfers (such as that described earlier for ARIES operations) because the primary concern is to minimize distance between the two vehicles and the path taken by the FV is of little concern provided that vehicle-to-vehicle collisions are prevented. The navigation requirements to complete a relaxed follow-the-leader behavior are less strict than those for track following because it is not necessary to duplicate the path taken by the leader. Thus relaxed following is a better choice for wider variety of vehicles with varying navigation accuracies.

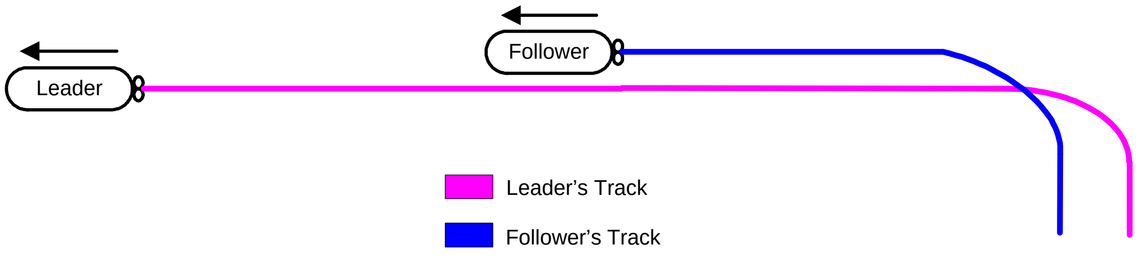


Figure 34. Relaxed following requires the FV to stay within a given min/max range from the LV while loosely following the LV's track.

C. PROBLEM STATEMENT

AUV dead reckoning errors quickly accumulate, typically at a rate of one to three percent of the distance traveled if the vehicle is not equipped with a highly accurate IMU. New integrated IMU/DVL systems, such as the Kearfott SEADeViL, are capable of calculating DR positions with errors as little as 0.05 %DT (Kearfott 2001), but such systems are typically prohibitively expensive so many AUV designers use cheaper less accurate systems.

It is important to minimize the relative navigation errors between vehicles performing cooperative behaviors because in many cases the vehicles base their actions on the estimated positions of the other vehicles in the group. Without a means of estimating and correcting relative errors, the accumulated DR errors will progressively degrade the accuracy of the behaviors and may lead to vehicle-to-vehicle collisions or loss of communications.

Figure 35 depicts a simple example demonstrating the impact of accumulated DR errors on cooperative behaviors that rely on the estimated positions of the other vehicles in the group. In this figure the two AUVs are attempting to perform track/path following using vehicle-to-vehicle communications. The LV’s DR-based position estimate indicates that the AUV is located on the intended track, but the LV’s true position is actually 10m south of its estimated position. During the follow-the-leader behavior the LV acoustically transmits its erroneous estimated position to the FV for use in determining its path. According to the FV’s DR solution it is located on the intended track, but it is really located 10m north of the track. So based on the erroneous position estimates the FV is on the same track as the LV, but in reality the FV’s true track is 20m north of the LV’s true track.

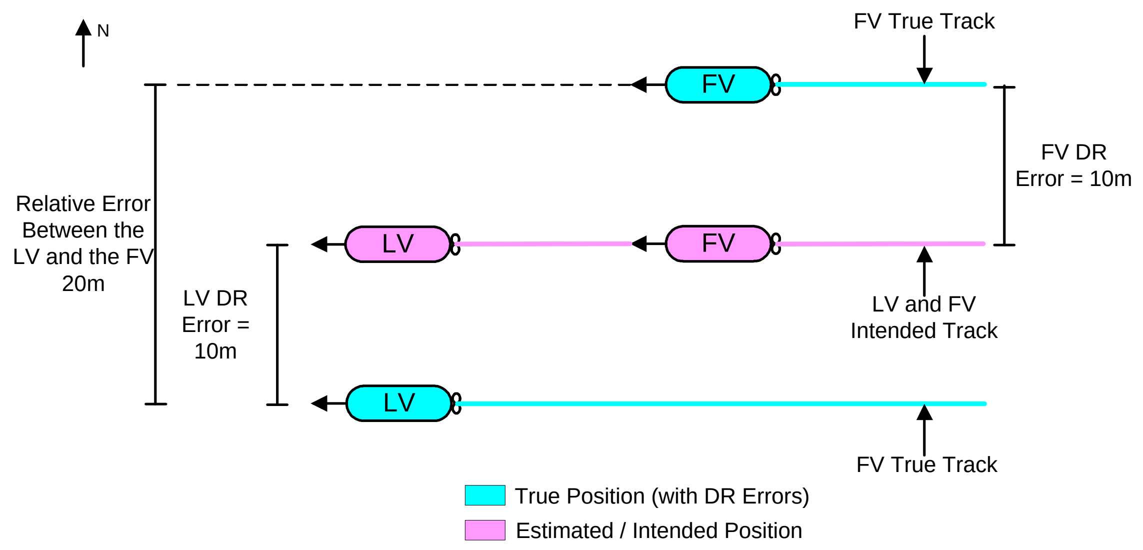


Figure 35. Two vehicles performing track/path following based on position estimates obtained using vehicle-to-vehicle communications will result in a total track following error equal to the sum of the errors for the two vehicles.

These errors can be greatly reduced if the vehicles share a common LBL navigation system or if they perform frequent GPS pop-ups. Some LBL systems can provide simultaneous position updates for all vehicles in the group every three seconds (Bernstein 2002). So excluding drop-outs and erroneous/noisy LBL fixes, the DR errors accumulate over three seconds and then the DR solution is reset to match the position provided by the LBL system.

Vehicles that do not share a common means of periodically correcting their accumulated DR errors typically have significant relative navigation errors, which cannot be automatically corrected without the use of specialized sensors or behaviors designed to correct these errors.

D. OBJECTIVES

The overall objective of the proposed procedure is to minimize the relative-navigation errors between two vehicles. The following constraints and objectives have been included to provide a solution that can be used with the maximum number of vehicle types/configurations and to support the long-term goals of cooperative behaviors.

1. Provide a solution that can support the follow-the-leader behavior and the development of future cooperative behaviors.

2. Minimize the communications and range readings required to complete the procedure, in order to maximize covertness and to support large groups of AUVs performing cooperative behaviors.
3. Minimize the time required to complete the correction behavior, in order to maximize the time available to complete the primary mission.
4. Avoid disrupting the LV's mission by having the FV preferentially perform the necessary maneuvers to collect the ranging data.
5. Provide a solution that does not require access to (or modification of) the vehicle's onboard software, so that the algorithms can be used on vehicles that operate using proprietary software.
6. Provide a solution that does not require either of the vehicles to be equipped with an expensive highly accurate DR navigation system.
7. Use standard functionality included with most commercial-off-the-shelf (COTS) acoustic modems, namely vehicle-to-vehicle communications and ranging readings. This approach eliminates the need for high-cost specialized sensors or modems.
8. Provide a solution that can be used periodically during relaxed follow-the-leader tracking to correct the errors that accumulate during the execution of the cooperative behavior.
9. Do not rely on a priori knowledge of the mission, which allows the LV to be retasked during the mission without the need to reprogram the FV.
10. Develop the methodology and algorithms to provide a better-than-linear reduction of the time required to complete missions using multiple AUVs and cooperative behaviors, thereby providing better efficiency than the traditional divide-and-conquer approach.

E. APPLICATIONS

The proposed solution is designed to support the following cooperative behaviors.

1. In-transit high-speed data transfers, where one vehicle follows a second vehicle at close range to perform high-speed data downloads. For example, the use of ARIES to download data from one or more REMUS vehicles, while each REMUS continues its mission without disruption.
2. Cooperative surface navigation fixes, where one vehicle in the group is equipped with a GPS receiver and it performs periodic GPS pop-ups to obtain position fixes. The vehicle then submerges and corrects the accumulated dead reckoning errors of the other vehicles in the group.
3. Cooperative submerged navigation fixes, where one vehicle in the group is equipped with a LBL receiver (or high resolution IMU) and this vehicle corrects the navigation errors of the other vehicles in the group.
4. Cooperative alignment of multiple LBL systems, where each vehicle is operating using a different LBL system and the proposed algorithm is used to correct the differences in the LBL systems.
5. Cooperative search, where multiple AUVs equipped with side-scan sonar perform minefield search operations and utilize the proposed algorithm to ensure proper spacing between the vehicles.

F. PROPOSED SOLUTION

1. Overview

The proposed solution uses the standard functionality of COTS acoustic modems, which provide vehicle-to-vehicle communications and a means of determining the distance between two vehicles (range readings). Using information obtained from the LV, the FV performs a set of maneuvers relative to the LV's position to collect a series of range readings. The collected data is then used to triangulate the position of the LV relative to the FV using the same methods discussed in Chapter III, except in this case the range readings are obtained using the FV instead of using transponders. The procedure and algorithms are designed to operate without a priori information regarding the mission or the LV’s intended track, which allows for additional mission flexibility. In this way, if the LV is re-tasked during the mission, the FV can still perform the navigation error-correction procedure without the need for reprogramming.

2. Assumptions

The proposed solution makes the following assumptions:

1) The vehicles accurately know their depth. This assumption simplifies the mathematics because it is only necessary to calculate a two-dimensional position correction factor. This is a practical assumption because all AUVs are equipped with depth sensors that are quite accurate and any depth-estimation errors resulting from environmental factors will be consistent for both vehicles since they are operating in close proximity.

2) The calibration is completed while the leader is transiting a long straight leg to minimize the amount of time between sample points. This is necessary because large accumulated errors between the data-sampling points will degrade the accuracy of the correction algorithm.

3) The FV can request status reports from the LV, which are referred to as LV updates. LV updates contain the LV’s current position, course, and speed.

3. Range Data Collection

a. Overview

The proposed algorithm uses acoustically communicated range values and position information collected at a series of points to correct the relative navigation errors between two vehicles. The data collection points, which are referred to as test points (TPs), are positioned relative to the LV’s position. The TP positions are fixed points in the LV’s local reference frame, and therefore their positions rotate and translate with the LV’s course and estimated position.

The FV can be preprogrammed with the position of the TPs. During the programming process it is assumed that the LV is located at the origin of the NED reference frame and pointing northward. Then while performing the navigation error correction behavior, the FV will rotate and translate the preprogrammed TPs to match the

LV’s position and course. Using this approach does not require the FV to be preprogrammed with the LV’s mission plan since the FV makes the appropriate TP position adjustments to account for the LV’s position and course during the mission. Alternatively, the FV can determine the location of the TPs during the mission based on a set of rules to ensure the points are spaced appropriately to ensure observability.

During the execution of the relative navigation error correction behavior, the FV maneuvers to each TP to collect range readings. For example, if ARIES performs an in-transit high-speed data download while REMUS is performing minefield search operations, REMUS serves as the LV and ARIES serves as the FV. The data-collection procedure is designed to minimize the impact on the LV’s mission, so in this example ARIES maneuvers to the TPs to collect the required data while REMUS continues its search mission.

b. Preprogrammed Test Points (TPs)

The preprogrammed test point TP approach requires the FV to be preprogrammed with a set of TPs prior to deployment. During the programming process it is assumed that the LV is at the origin of the NED reference frame and pointing northward, as depicted in Figure 36. While the mission is underway, the FV will rotate and translate the programmed points to account for the LV’s position and course using Equations 12 and 13.

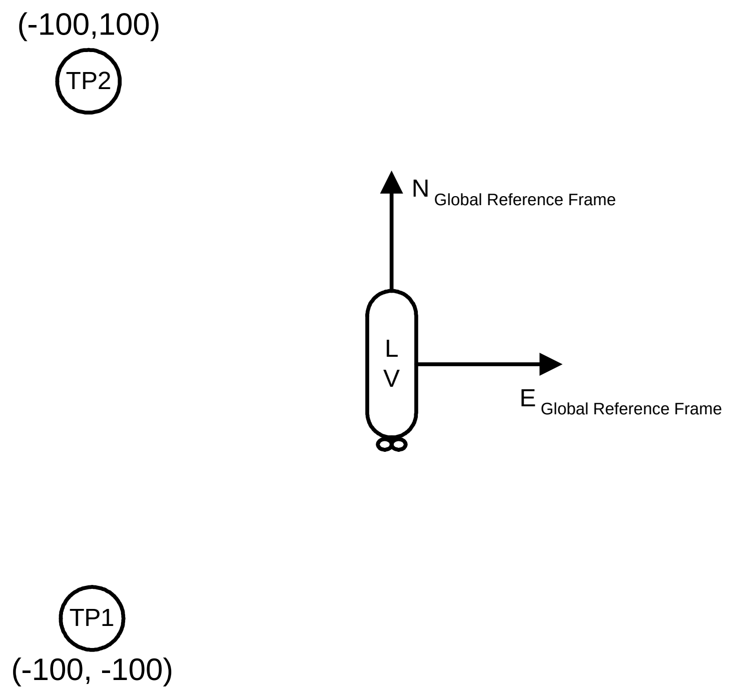


Figure 36. Test point programming for the parallel track correction configuration where the FV’s track is parallel to the LV’s track.

$$\left[TPi_{Rot/N} \quad TPi_{Rot/E} \quad TPi_{Rot/D} \right] = \left[TPi_{N} \quad TPi_{E} \quad TPi_{D} \right] * \begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 12: Rotation of preprogrammed test point i to account for the LV’s current course.

where,
$TPi_{Rot/N}$, $TPi_{Rot/E}$, $TPi_{Rot/D}$ = The north, east, down position of test point i after a rotation of θ degrees.
TPi_{N}, TPi_{E}, TPi_{D} = The north, east, and down components of test point i as defined during the preprogramming of the FV (e.g. as defined in Figure 36)
θ = The desired angle of rotation, i.e. the LV’s course.

$$\begin{bmatrix} TPx_{Cor/N} & TPx_{Cor/E} & TPx_{Cor/D} \end{bmatrix} = \begin{bmatrix} LV_N & LV_E & LV_D \end{bmatrix} + \begin{bmatrix} TPx_{Rot/N} & TPx_{Rot/E} & TPx_{Rot/D} \end{bmatrix}$$

Equation 13: Translation of test point i for to account for the LV's current position.

where,

$TPi_{Cor/N}$, $TPi_{Cor/E}$, $TPi_{Cor/D}$ = The north, east, and down components of test point i after the rotation and translation to compensate for the LV's position and rotation.

LV_N, LV_E, LV_D = The north, east, and down components of the LV's current position obtained from vehicle-to-vehicle communications.

$TPi_{Rot/N}$, $TPi_{Rot/E}$, $TPi_{Rot/D}$ = The north, east, down position of test point i after a rotation of θ degrees. These values are obtained from Equation 12.

At the start of the relative navigation-error correction procedure the FV acoustically sends a request instructing the LV to transmit a LV update. It is important to note that the position returned by the LV will be its estimated position and it will not accurately reflect its true position. Then using the information contained in the LV update and equations 10 and 11, the FV rotates and translates test point one to compensate for the LV's course and position. The FV will the transit to test point 1 while using the LV's course and speed information contained in the LV update to continuously update the estimated position of the TP.

After reaching the approximate location of test point 1, the FV requests another LV update. Using the information contained in the LV update the FV updates the position of test point 1 using equations 12 and 13, and fine-tune its position to match the new test point 1 position. Alternatively, the FV may use the LV update to calculate its actual position relative to the LV using Equation 14 instead of fine-tuning its position. The FV will then match the speed and course of the LV in order maintain its position relative to the LV. Next the FV uses its acoustic modem to determine its range to the LV, which is stored along with the FV's position relative to the LV for later processing. The FV will then repeat the above procedure to collect the data for test point 2.

$$\begin{bmatrix} TPi_N & TPi_E & TPi_D \end{bmatrix} = \begin{bmatrix} FV_N & FV_E & FV_D \end{bmatrix} - \begin{bmatrix} LV_N & LV_E & LV_D \end{bmatrix}$$

Equation 14: Test point position calculation based on the current positions of the LV and FV.

where,
TPi_N, TPi_E, TPi_D = North, east, and down components of test point i.
FV_N, FV_E, FV_D = North, east, and down components of the FV’s estimated position in the NED reference frame.
LV_N, LV_E, LV_D = North, east, and down components of the LV’s estimated position in the NED reference frame.

c. Dynamically Determined Test Points (TPs)

It is possible to utilize the proposed algorithm without pre-programming the locations of the test points. This approach requires the FV to select the test points dynamically during the mission based on information contained in the LV updates and a set of rules to ensure observability. During the mission the FV requests a LV update, calculates the position of the desired test point, and transits to the point while continuously updating the position of the TP based on the course and speed of the LV. After reaching the estimated TP position, the FV will then request another LV update and calculate its position relative to the LV’s current position using Equation 14. While maintaining its relative position to the LV, the FV then request a range reading. The FV stores its position relative to the LV and the ranging information for the later use when calculating the navigation error correction factor. The FV then repeats the procedure to collect data for a second test point.

d. Test Point (TP) Configurations

There are several test point configurations that can be successfully used to correct the relative navigation errors. The first configuration allows the LV and FV to perform the correction procedure while transiting on parallel tracks, such as shown in Figures 37, 38, 39, and 40. Figures 41 and 42 depict a second configuration where the test points are located behind the LV, which can be performed periodically while the LV and the FV are executing the relaxed follow-the-leader behavior. Furthermore, the approach depicted in Figures 40 and 41 requires less energy than the parallel track method because it is more costly, in terms of energy consumption, to have the FV “sprint” ahead of the LV. It is important to note that the presented equations assume that test point 2 is the origin of the triangulation reference frame, which results in the triangulation reference frames depicted in Figures 37 through 42.

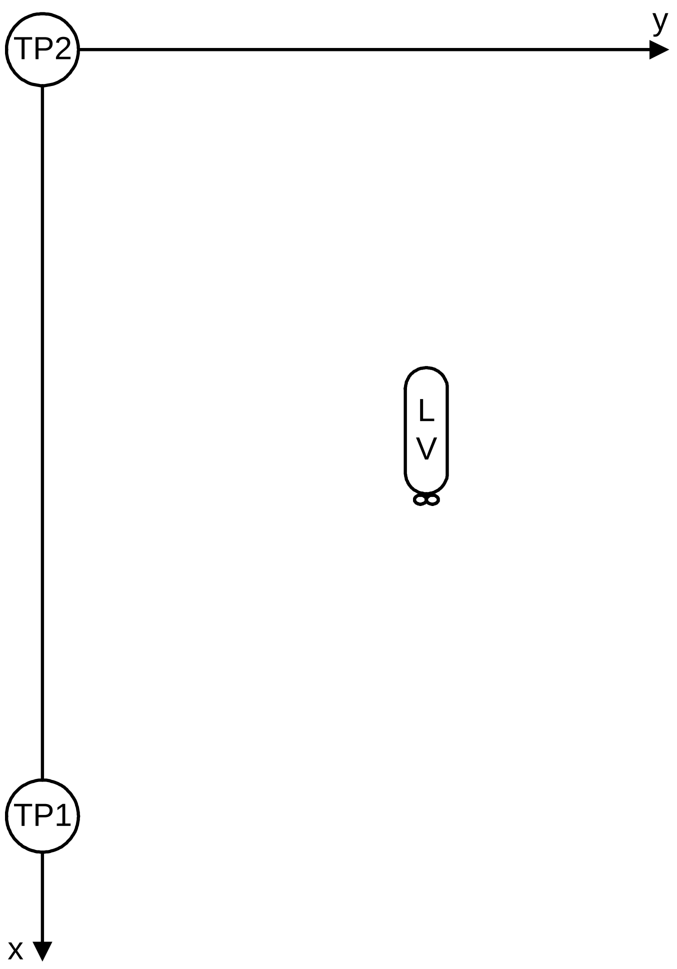


Figure 37. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the port side of the vehicle and TP1 is behind the LV with TP2 as the origin of the reference frame.

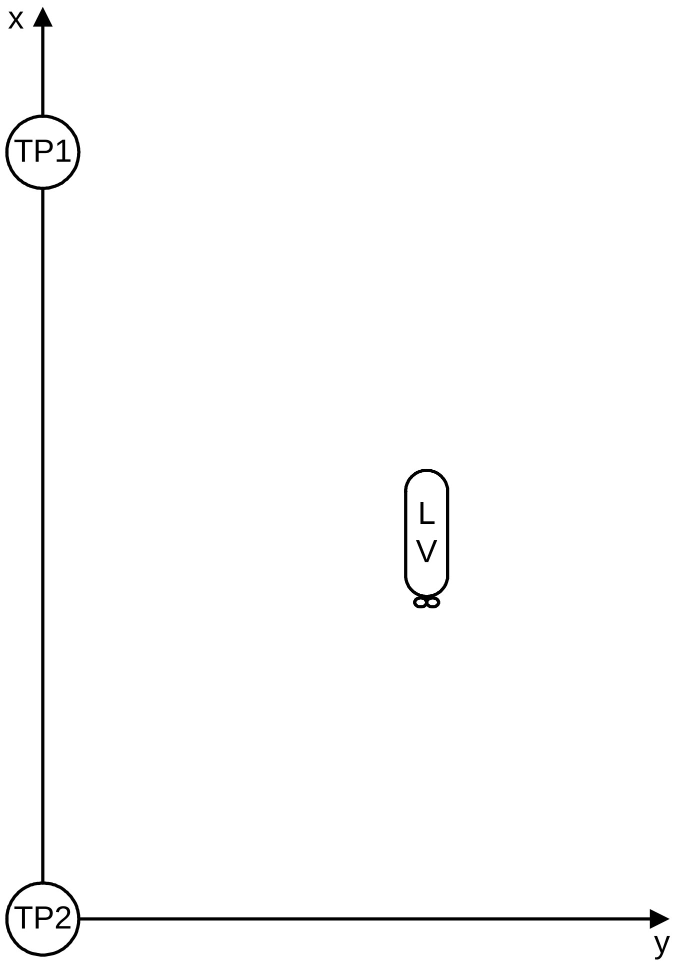


Figure 38. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the port side of the vehicle and TP1 is ahead of the LV with TP2 as the origin of the reference frame.

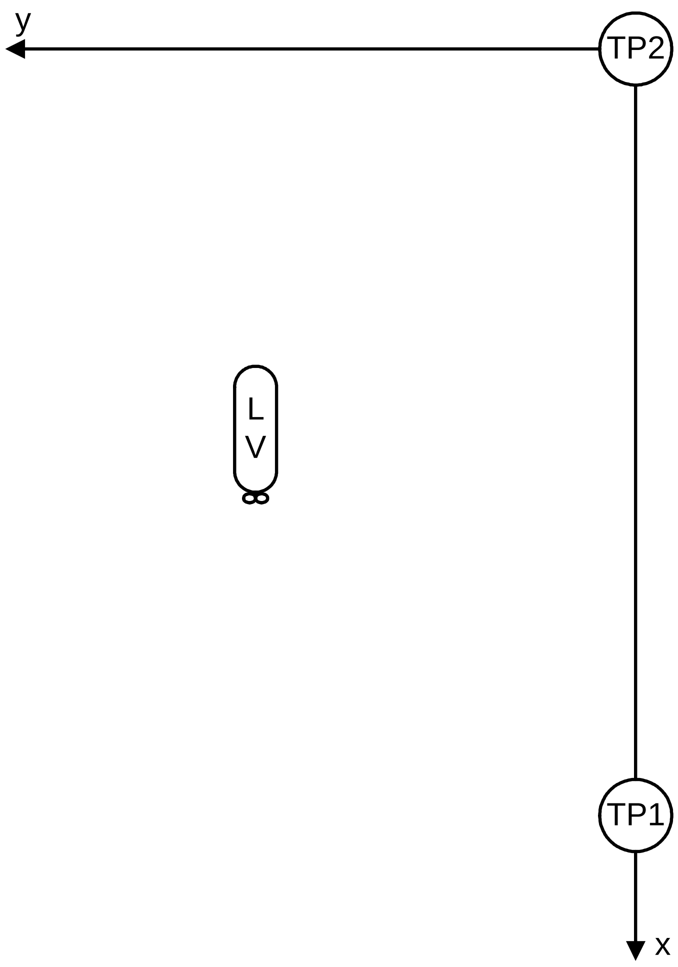


Figure 39. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the starboard side of the vehicle and TP1 is behind the LV with TP2 as the origin of the reference frame with TP2 as the origin of the reference frame.

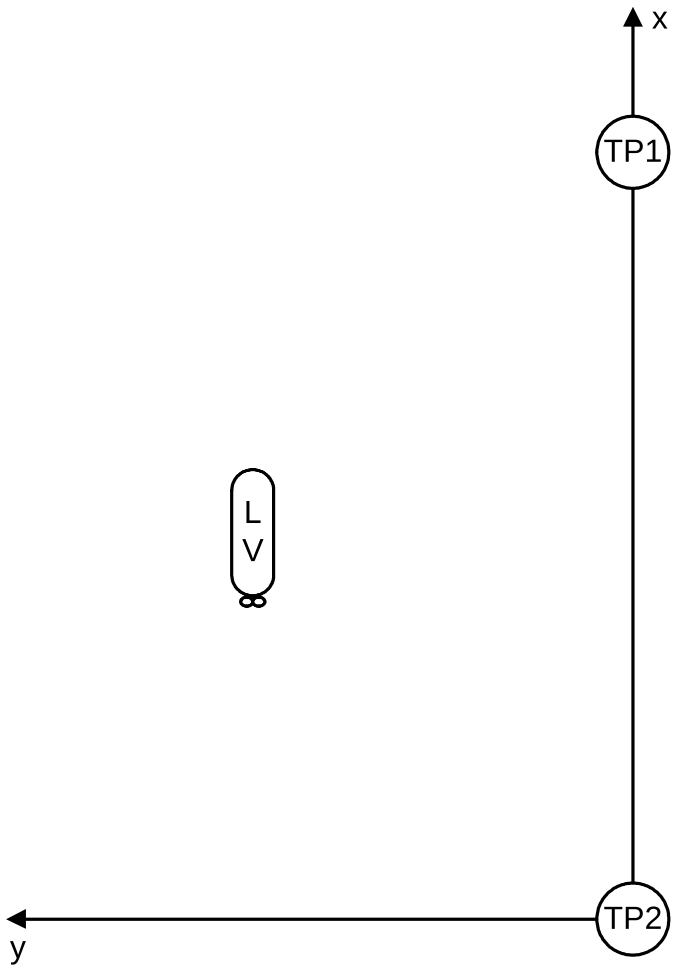


Figure 40. Triangulation reference frame for the parallel track test point configuration where the TPs are located on the starboard side of the vehicle and TP1 is ahead of the LV with TP2 as the origin of the reference frame.

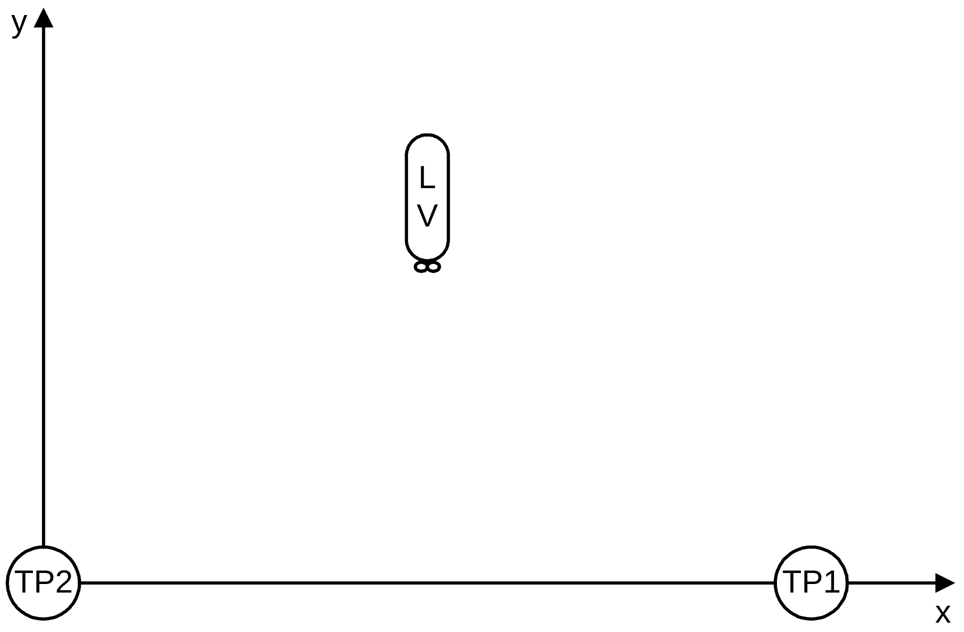


Figure 41. Triangulation reference frame for the follow-the-leader test point configuration where TP1 is located on the starboard side of the LV with TP2 as the origin of the reference frame.

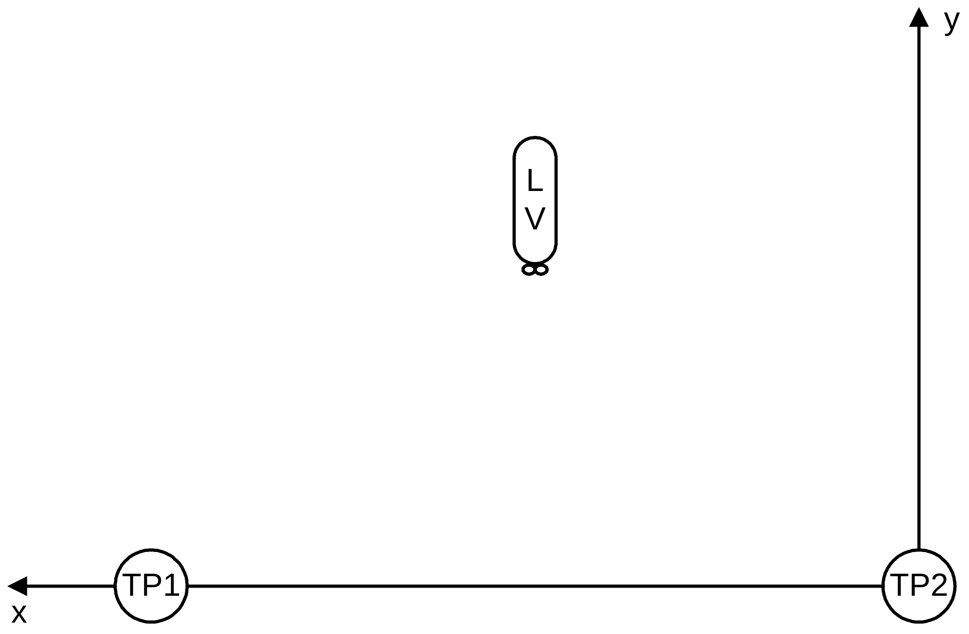


Figure 42. Triangulation reference frame for the follow-the-leader test point configuration where TP1 is located on the port side of the LV with TP2 as the origin of the reference frame.

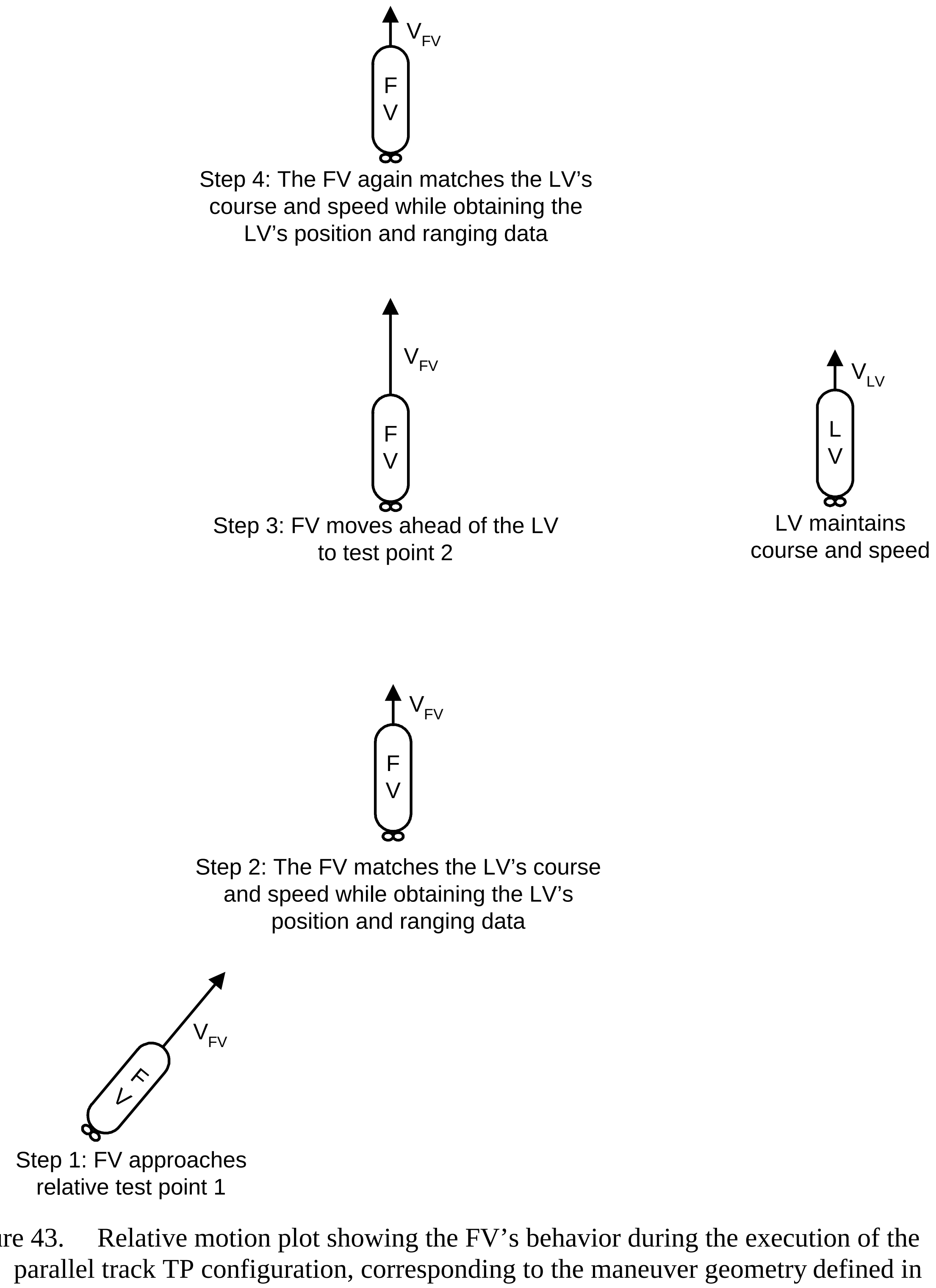

V_FV
F V
Step 4: The FV again matches the LV's course and speed while obtaining the LV's position and ranging data
V_FV
F V
Step 3: FV moves ahead of the LV to test point 2
V_LV
L V
LV maintains course and speed
V_FV
F V
Step 2: The FV matches the LV's course and speed while obtaining the LV's position and ranging data
V_FV
F V
Step 1: FV approaches relative test point 1

Figure 43. Relative motion plot showing the FV's behavior during the execution of the parallel track TP configuration, corresponding to the maneuver geometry defined in Figure 37.

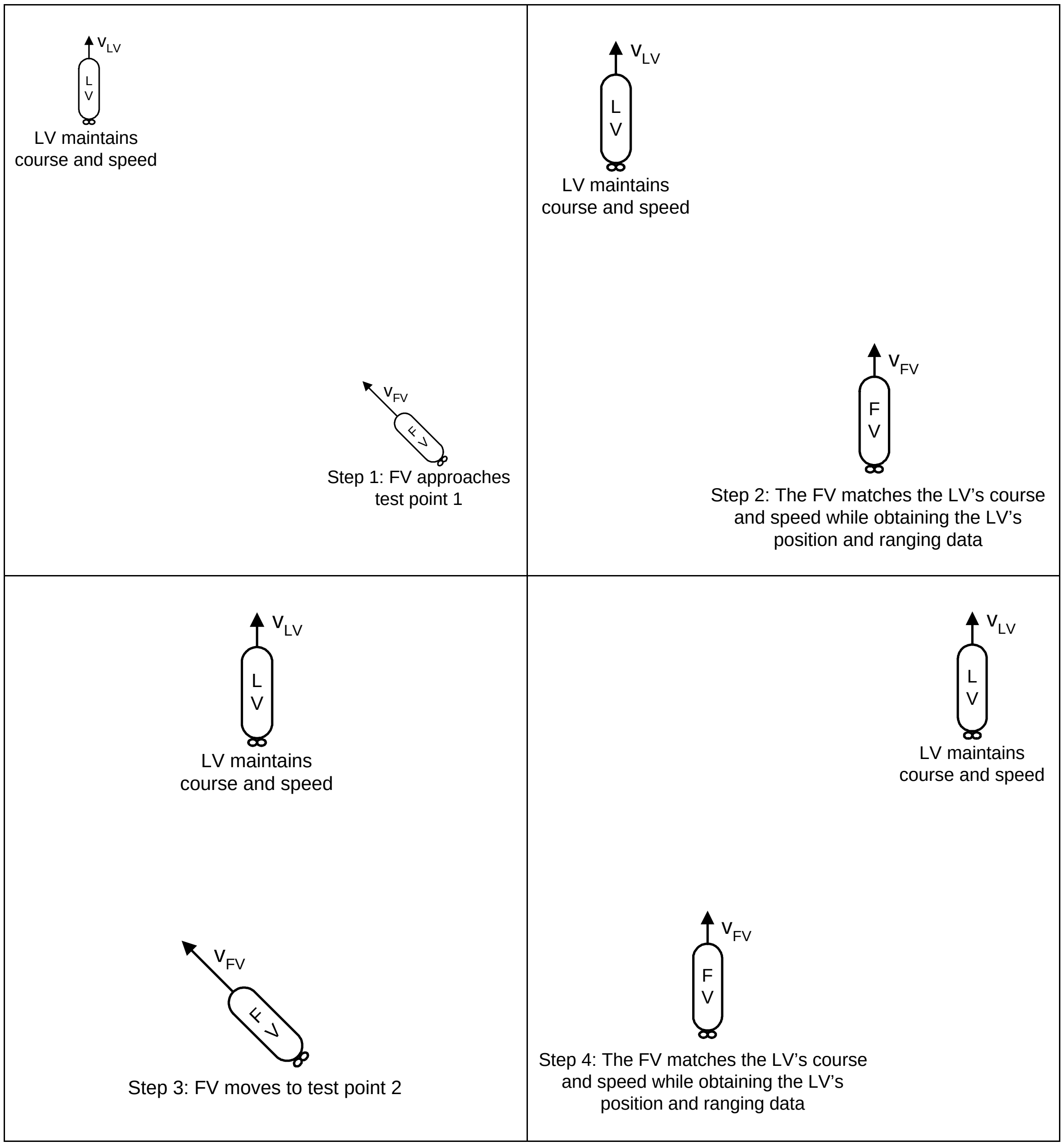

v_LV
LV
LV maintains course and speed
v_FV
FV
Step 1: FV approaches test point 1
v_LV
LV
LV maintains course and speed
v_FV
FV
Step 2: The FV matches the LV's course and speed while obtaining the LV's position and ranging data
v_LV
LV
LV maintains course and speed
v_FV
FV
Step 3: FV moves to test point 2
v_LV
LV
LV maintains course and speed
v_FV
FV
Step 4: The FV matches the LV's course and speed while obtaining the LV's position and ranging data

Figure 44. Relative motion plot showing the FV's behavior during the execution of the follow-the-leader TP configuration, corresponding to the maneuver geometry depicted in Figure 41.

4. Relative Navigation Error Correction Factor Calculation

a. Overview

This section explains how to use the range data collected in the previous section to calculate the relative navigation-error correction factor, which is an offset that the FV can add to the LV's estimated position received via ACOMMS to minimize the relative navigation error between the two vehicles.

b. Correction Factor Calculation Using Two Points

The correction factor calculation using two points requires that the FV collect range values for two TPs, such as depicted in Figures 37 – 42. After the data collection process is complete, the FV will have the following information available to calculate the correction factor:

1. The range between the LV and FV while the FV is located at each TP. It is important to note that the range values reflect the true distance between the vehicles, whereas the test point positions are based on DR estimates.
2. The estimated position of the TPs relative to LV.
3. The estimated positions of the LV and the FV when the ranging data is collected for each TP.

The FV then uses the estimated position of the TPs relative to the LV to establish an artificial triangulation reference frame, as discussed in Chapter III and shown in Figures 37 – 42. The ranging information collected at each test point is then used to triangulate the position of the vehicle in the triangulation frame using Equations 15, 16, and 17. The LV's position in the correction frame is then rotated to match the orientation of the NED reference frame. The equations used to perform the rotation are dependent on the test point configuration used. If the test point configurations depicted in Figures 37, 40, or 41 are used, then Equations 18 and 19 are used to rotate the triangulation reference frame. If the test points are configured as indicated in Figures 38, 39, or 42, then Equations 20 and 21 are used to rotate the triangulation reference frame. Next, the FV calculates the estimated position of the LV relative to the test point 2 using Equation 22. Using Equation 23 the correction factor is then calculated based on the DR estimated position of the LV relative to test point 2 and triangulated position of the LV relative to test point 2. The LV can then uses the correction factor as shown in Equation 20 to correct the relative navigation errors between the LV and the FV.

$$L_{baseline} = \sqrt{(TP2_N - TP1_N)^2 + (TP2_E - TP1_E)^2}$$

Equation 15: Calculation of the baseline length, which is the distance between test points 1 and 2.

where,
$L_{baseline}$ = the baseline length, which is the distance between test points 1 and 2.
$TP1_N$, $TP1_E$ = The north and east components of test point 1's position in the NED reference frame.
$TP2_N$, $TP2_E$ = The north and east components of test point 2's position in the NED reference frame.

$$x_{vehicle} = \frac{R_{TP2}{}^2 + L_{baseline}{}^2 - R_{TP1}{}^2}{2L_{baseline}}$$

Equation 16: Calculation of the vehicle's x-position in the triangulation reference frame using the ranging data obtained from two test points (McTrusty 2000).

where,
$x_{vehicle}$ = the x-position of the LV in the triangulation reference frame.
R_{TP1} = The range between the LV and test point 1.
R_{TP2} = The range between the LV and test point 2.
$L_{baseline}$ = The distance between transponders 1 and 2 as defined in Equation 15.

$$y_{vehicle} = \sqrt{R_{TP2}{}^2 - x_{vehicle}{}^2}$$

Equation 17: Calculation of the vehicle's y-position in the triangulation reference frame using ranging information and the x-position calculated using Equation 16 (McTrusty 2000).

where,
$y_{vehicle}$ = the y-position of the LV in the triangulation reference frame.
$x_{vehicle}$ = the x-position of the vehicle in the triangulation reference frame calculated using Equation 16.
R_{TP2} = The range between the LV and test point 2.

$$q_{x\,\mathrm{Re}\,lToE} = a\tan 2((TP1_N - TP2_N),(TP1_E - TP2_E))$$

Equation 18: Calculation of the angle between the triangulation frame’s x-axis and the NED east axis, for use with the TP configurations depicted in Figures 37, 40 and 41.

where,

θ$_{xRelToE}$ = The angle between the correction/triangulation frame’s x-axis and the NED reference frame’s east axis (as shown in Figure 29).

TP1$_N$, TP1$_E$ = The north and east components of test point 1’s position in the NED frame.

TP2$_N$, TP2$_E$ = The north and east components of test point 2’s position in the NED frame.

$$\begin{bmatrix} LV_{E/triangulation/TP2} & LV_{N/triangulation/TP2} & 1 \end{bmatrix} = \begin{bmatrix} x_{vehicle} & y_{vehicle} & 1 \end{bmatrix} * \begin{bmatrix} \cos q_{x\,\mathrm{Re}\,lToE} & \sin q_{x\,\mathrm{Re}\,lToE} & 0 \\ -\sin q_{x\,\mathrm{Re}\,lToE} & \cos q_{x\,\mathrm{Re}\,lToE} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 19: Rotation of triangulation reference frame to match the NED reference frame, for use with the TP configurations depicted in Figures 37, 40 and 41 (Healey 2003).

where,

LV$_{N/triangulation/TP2}$, LV$_{E/triangulation/TP2}$ = the triangulated position of the LV relative to test point 2. Note that the east and north order indicated in the equation is intentional to account for the triangulation frame’s x and y axes being aligned with the NED east and north axes respectively. This approach eliminates the need for a second rotation, therefore reducing the CPU time required to perform the conversion.

x$_{vehicle}$, y$_{vehicle}$ = The X and Y components of the LV’s position in the triangulation frame (from Equations 16 and 17).

θ$_{xRelToE}$ = The angle between the correction/triangulation frame’s x-axis and the NED frame’s east axis (from Equation 18).

$$q_{x\,\mathrm{Re}\,lToE} = a\tan 2((TP1_N - TP2_N),(TP1_E - TP2_E))$$

Equation 20: Calculation of the angle between the triangulation frame’s x-axis and the NED east axis, for use with the TP configurations depicted in Figures 38, 39, and 42.

where,

θ$_{xRelToE}$ = The angle between the correction/triangulation frame’s x-axis and the NED reference frame’s east axis (as shown in Figure 30).

TP1$_N$, TP1$_E$ = The north and east components of test point 1’s position in the NED frame.

TP2$_N$, TP2$_E$ = The north and east components of test point 2’s position in the NED frame.

$$\begin{bmatrix} LV_{N/triangulation/TP2} & LV_{E/triangulation/TP2} & 1 \end{bmatrix} = \begin{bmatrix} x_{vehicle} & y_{vehicle} & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta_{x\mathrm{Re}lToE} & \sin\theta_{x\mathrm{Re}lToE} & 0 \\ -\sin\theta_{x\mathrm{Re}lToE} & \cos\theta_{x\mathrm{Re}lToE} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation 21: Rotation of triangulation reference frame to match the NED reference frame, for use with the TP configurations depicted in Figures 38, 39, and 41 (Healey 2003).

where,
$LV_{N/triangulation/TP2}$, $LV_{E/triangulation/TP2}$ = the triangulated position of the LV relative to test point 2.
$x_{vehicle}$, $y_{vehicle}$ = The X and Y components of the LV's position in the triangulation frame (from Equations 16 and 17).
$\theta_{xRelToE}$ = The angle between the correction/triangulation frame's x-axis and the NED frame's east axis (from Equation 20).

$$\begin{bmatrix} LV_{N/est/TP2} & LV_{E/est/TP2} \end{bmatrix} = \begin{bmatrix} LV_{N/est} & LV_{E/est} \end{bmatrix} - \begin{bmatrix} TP2_{N/est} & TP2_{E/est} \end{bmatrix}$$

Equation 22: Calculation of the LV's estimated position relative to test point 2 based on DR position estimates.

where,
$LV_{N/est/TP2}$, $LV_{E/est/TP2}$ = The LV's position relative to test point 2 based on DR estimated positions.
$LV_{N/est}$, $LV_{E/est}$ = The LV's estimated DR position in the NED reference frame while the ranging data was collected at test point 2.
$TP2_{N/est}$, $TP2_{E/est}$ = The estimated position of test point 2 in the NED reference frame at the time that the range data was received.

$$\begin{bmatrix} CF_N & CF_E \end{bmatrix} = \begin{bmatrix} LV_{N/est/TP2} & LV_{E/est/TP2} \end{bmatrix} - \begin{bmatrix} LV_{N/triangulation/TP2} & LV_{E/triangulation/TP2} \end{bmatrix}$$

Equation 23: Calculation of the correction factor used to minimize the relative navigation errors between the LV and the FV.

where,
CF_N, CF_E = the north and east components of the correction factor, which can be added to all future LV positions received by the FV to minimize the relative navigation errors.
$LV_{N/est/TP2}$, $LV_{E/est/TP2}$ = The LV's position relative to test point 2 based on DR estimated positions (from Equation 22).
$LV_{N/triangulation/TP2}$, $LV_{E/triangulation/TP2}$ = the triangulated position of the LV relative to test point 2 (from Equation 19 or 21).

$$\left[LV_{N/corrected} \quad LV_{E/corrected}\right] = \left[LV_{N/est} \quad LV_{E/est}\right] + \left[CF_N \quad CF_E\right]$$

Equation 24: The correction of the LV's position to minimize the relative-navigation errors between the LV and FV.

where,
$LV_{N/corrected}$, $LV_{E/corrected}$ = The position of the LV in the NED reference frame after applying the correction factor to correct the relative navigation errors between the LV and the FV.
$LV_{N/est}$, $LV_{E/est}$ = The LV's estimated DR position as received in LV updates (in the NED reference frame).
CF_N, CF_E = the north and east components of the correction factor (from Equation 23).

c. Correction Factor Calculation Using Three or More Test Points

An alternative approach to the two test point correction factor calculation method presented in the previous section is to use three or more test points. This approach requires the FV to perform additional maneuvers so the TPs are not collinear in order to ensure observability. However, the three test point method is more robust because the baseline crossover issue associated with the use of two test points is not an issue. Therefore, it is advisable to use this approach after the initial rendezvous is completed. Afterwards, the two test point method can be use periodically to improve the correction factor with a reduced likelihood of the LV being positioned on the wrong side of the baseline as a result of large accumulated DR errors.

This approach requires that the FV collect range values for three or more test points, such as described in the previous section for the two test point approach. The FV then calculates the correction factor by setting up a set of equations based on Equation 25. Equation 25 uses the range data and the positions of the TPs to calculate an offset, which represents the navigation error between the LV and the FV.

$$R_i = \sqrt{(TP_{N/i} - CF_N)^2 + (TP_{E/i} - CF_E)^2 + (TP_{D/i} - LV_D)^2}$$

Equation 25: The calculation of the relative navigation error correction factor using three or more test points.

where,
i = 1, 2, …, N, where N is the number of range readings used.
R_i = the range reading obtained for test point i.
$TP_{N/i}$, $TP_{E/i}$, $TP_{D/i}$ = the north, east, and down components of test point i's position relative to the LV based on Equation 14.
CF_N, CF_E = the north and east components of the correction factor used to correct the relative navigation errors between the LV and the FV.
LV_D = The LV's depth at the time the data was collected for test point i.

G. SUMMARY

AUVs that do not share a common means of correcting their accumulated DR errors, such as LBL or GPS, often have significant relative errors that will degrade the accuracy of cooperative behaviors and may lead to vehicle-to-vehicle collisions. Through the use of ranging values obtained by the FV while located at positions relative to the LV and the triangulation techniques discussed in Chapter III, the relative navigation errors can be greatly reduced. The proposed algorithm is not computationally intensive, and therefore does not require significant CPU time to perform the calculations required to correct the relative-navigation errors between the vehicles. Furthermore, assuming one vehicle in the group is equipped with GPS or an LBL system, the same navigation correction techniques may be employed to correct the global navigation errors of all vehicles in the group.

V. MATLAB EVALUATION OF THE RELATIVE-NAVIGATION ERROR-CORRECTION ALGORITHM

A. INTRODUCTION

This chapter discusses the Matlab simulation of the relative-navigation error-correction algorithm presented in Chapter IV. The simulation allows the user to monitor or adjust a large number of parameters that might impact the performance of the algorithm, thereby allowing the performance of sensitivity analysis. The results of several experiments performed using the simulator are also presented.

B. OBJECTIVES

The objective of the Matlab simulation is to evaluate the performance of the proposed relative-navigation error-correction algorithm under varying conditions. Furthermore, the simulation provides a means of determining under what conditions the algorithm fails and the optimal test point configurations.

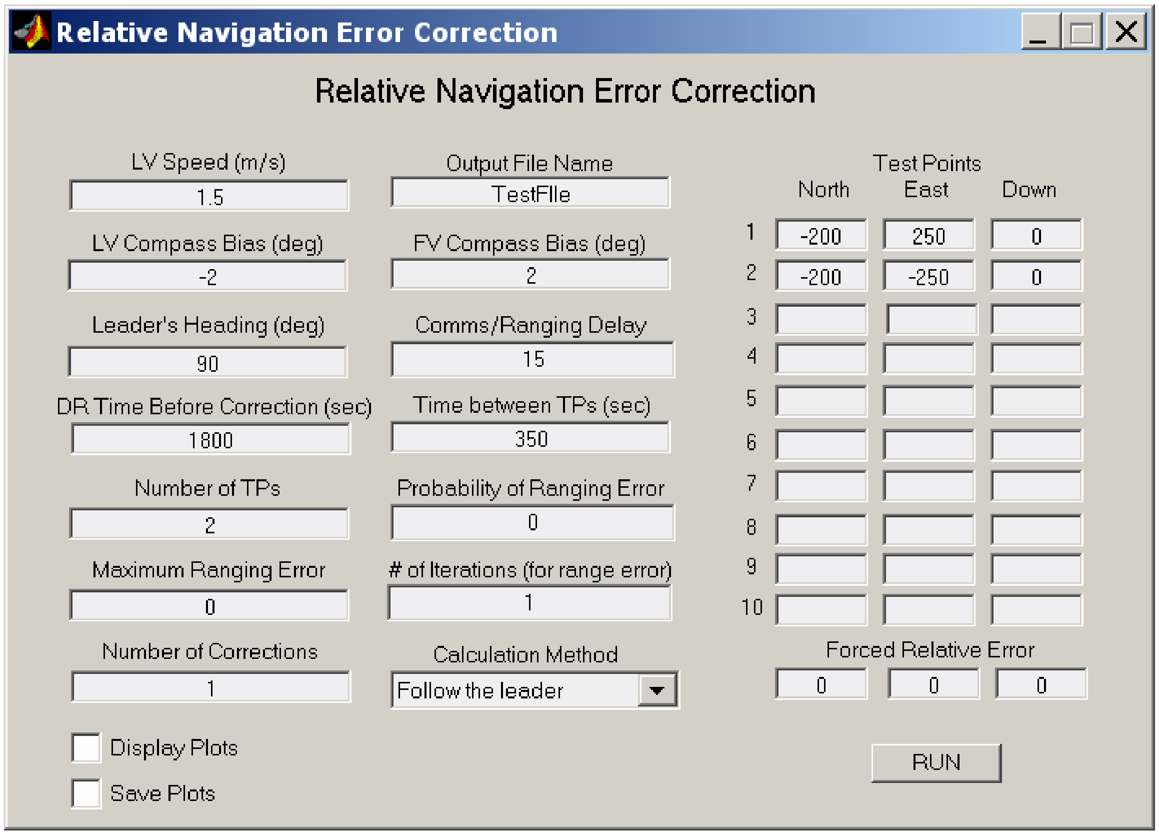
C. ASSUMPTIONS

While implementing the relative-navigation error-correction algorithm it was necessary to make the following assumptions.

1. Vehicle dynamics are not included in the simulation. However, realistic speeds were used.
2. The vehicle-to-vehicle communications are perfect. In the event of a drop-out or corrupted message, the FV simply requests the message again.
3. Future work needs to perform these experiments using simulated hydrodynamics with realistic communications losses and delay.

D. APPROACH

In keeping with the above-stated objectives, the simulator provides the maximum flexibility by allowing the user to monitor or adjust a large number of parameters that might impact the performance of the algorithm. Figure 45 shows the user modifiable parameters provided by the simulator's graphical user interface (GUI) and Table 1 provides an explanation of each field in the GUI.


Relative Navigation Error Correction

Relative Navigation Error Correction

LV Speed (m/s)	1.5
Output File Name	TestFIle
LV Compass Bias (deg)	-2
FV Compass Bias (deg)	2
Leader's Heading (deg)	90
Comms/Ranging Delay	15
DR Time Before Correction (sec)	1800
Time between TPs (sec)	350
Number of TPs	2
Probability of Ranging Error	0
Maximum Ranging Error	0
# of Iterations (for range error)	1
Number of Corrections	1
Calculation Method	Follow the leader

Test Points

	North	East	Down
1	-200	250	0
2	-200	-250	0
3			
4			
5			
6			
7			
8			
9			
10			
Forced Relative Error	0	0	0

☐ Display Plots

☐ Save Plots

Figure 45. The graphical user interface (GUI) for the Matlab-based relative navigation error correction simulation.

Field Name (from Figure 38)	**Description**
LV Speed	The speed of the LV before and during the calibration.
Output File Name	The file name for the generated comma delimited ASCII data, which can be opened in Excel or Matlab for later analysis.
LV Compass Bias	The LV's compass bias, which is used to calculate its DR errors.
FV Compass Bias	The FV's compass bias, which is used to calculate its DR errors.
LV Heading	The LV's heading before and during the correction.
Comms/Ranging Delay	The time elapsed between when the FV receives the LV update message and when the range is received.
DR Time Before Correction	The length of time that the LV used DR before the start of the correction. The DR time of the FV before the start of the correction is calculated based on the distance between (0, 0) and the position of TP1 after rotation and translation to adjust for the LV's position and orientation.
Time between TPs	The amount of time required to transit from test point 1 to test point 2.
Number of TPs	The number of test points to be used for the calibration.
Probability of ranging error	The probability of a spurious range reading.
Maximum ranging error	The maximum ranging error forced into the range reading in the event of a spurious reading. The ranging error are randomly selected between +/–(Max ranging error).
# of iterations (for range error)	The number of independent corrections to be performed. This function collects a data set to determine the effects of random ranging errors (as set using "Probablity of ranging error" and Maximum ranging error").
Number of Corrections	The number of times to perform the correction procedure so the effects of repeated corrections may be analyzed.
Calculation Method	Used to select the two test point follow-the-leader correction method, the parallel track method, or the system of equations approach that uses three or more TPs.
Test Points	The north, east, and down positions of the test points. The points are entered as if the LV is at the origin of the NED reference frame and pointing northward.
Forced Relative Error	The north, east, and down positions of the forced error. This function is used to analyze the performance of the algorithm in correcting the offset between two LBL systems.

Table 1 A description of the GUI fields for the Matlab simulation used to evaluate the relative navigation error correction algorithm.

The simulator includes a special automatic data collection function that is not indicated in the GUI. This function automatically varies the compass bias for each vehicle to obtain a data set representing the effects on varying compass biases. This function is activated by setting both the LV and FV's compass biases equal to -1. The simulation then performs 49 independent simulation runs, with the compass biases for each vehicle varied. The LV's compass bias is varied from 0 to -3 degrees in increments of -0.5 degrees. The FV's compass bias is varied from 0 to +3 degrees in increments of 0.5 degrees. A total of 49 independent simulation runs are completed where all compass bias combinations are tested.

The simulation is also capable of automatically generating approximate transit times required by the FV to travel from test point 1 to test point 2. This option is invoked by leaving the "Time between TPs" field blank.

E. ANALYSIS

1. Overview

This section discusses the simulation results and the performance of the relative-navigation error-correction algorithm presented in Chapter IV.

2. Varying the Time Required to Complete the Correction Procedure

The effects of varying the time required to complete the relative-navigation error-correction procedure were examined to determine the impact of navigation errors accumulated during the execution of the correction behavior. This test was completed using the follow-the-leader test point configuration depicted in Figure 46, with a LV speed of 1.5 m/s, the LV dead reckoned without LBL updates for 30 minutes before the start of the correction, and the times required to transit from TP1 and TP2 varied from 0 to 1000 seconds. Furthermore, the automatic data collection function was used to perform 49 independent simulation runs for each transit time. The results of this simulation are summarized in Figure 47.

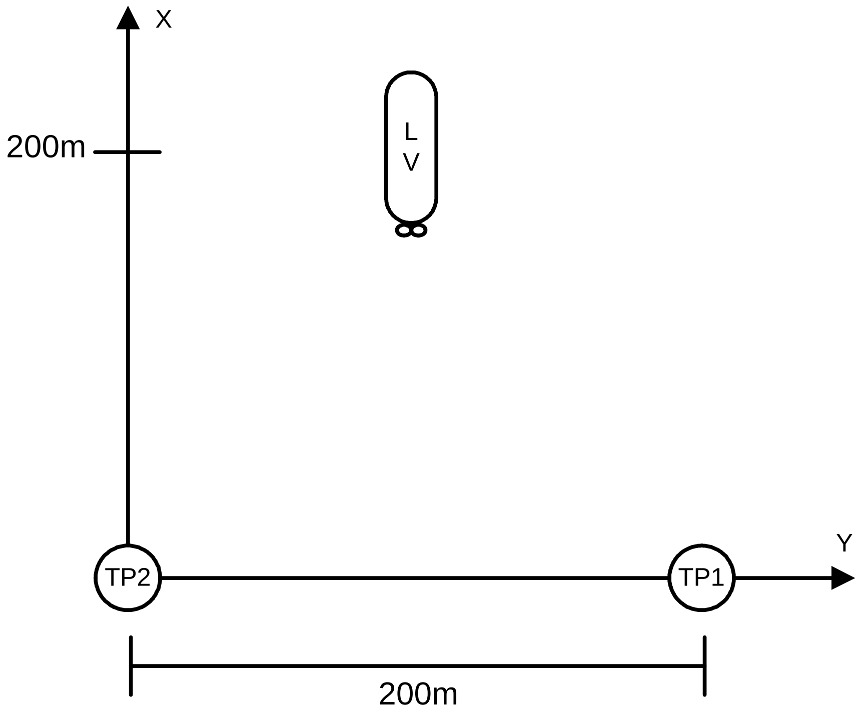


Figure 46. The test point configuration for evaluating the effects of relative errors accumulated during the relative-navigation error-correction procedure.

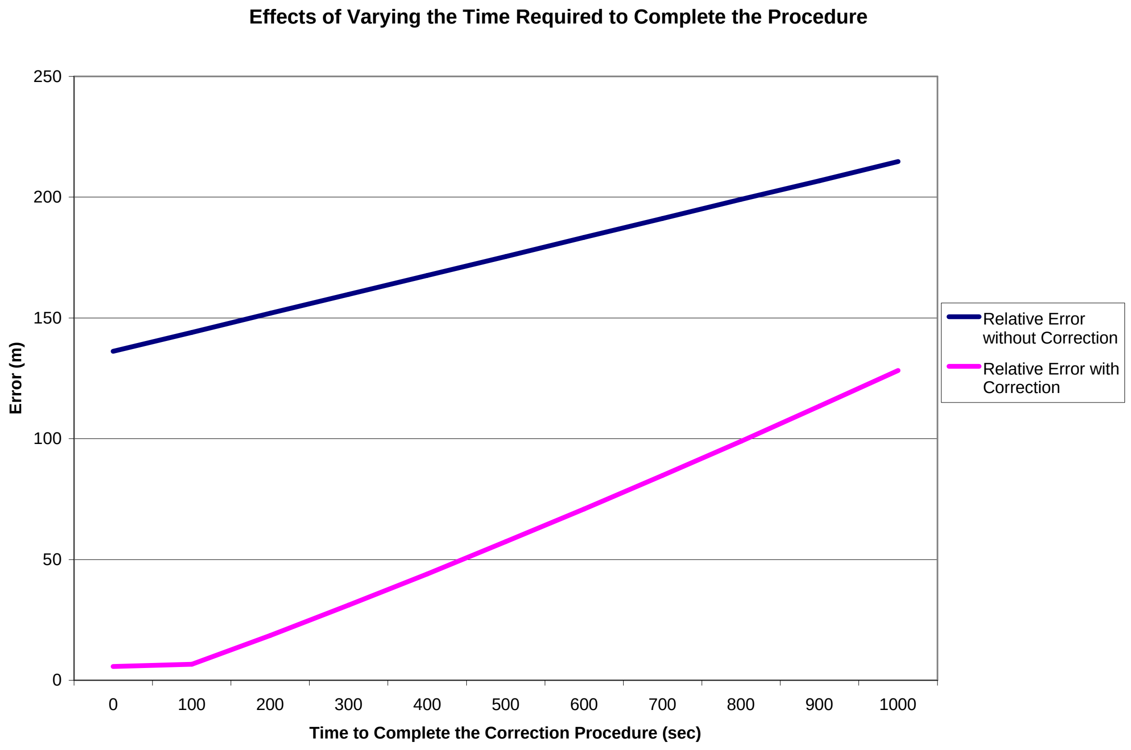


Figure 47. The results of varying the time required to complete the correction procedure with a baseline length of 200m, the baseline located 200m behind the LV, DR time of 1800 seconds before the start of the correction behavior, and compass biases determined by the automatic data collection function.

Based on the results of this experiment, the proposed relative-navigation error-correction procedure is extremely sensitive to relative-navigation errors accumulated during the execution of the correction behavior.

3. Varying Baseline Length

The impact of varying the baseline length was examined to determine its effects on the overall performance of the algorithm. The test used the follow-the-leader test point configuration depicted in Figure 48 with the baseline varied between 50 and 800 meters, a LV's speed of 1.5 m/s, and the vehicles used DR without LBL updates for 30 minutes before the start of the correction. The automatic data collection function was used, so the compass biases for the vehicles were varied as described above. The results of this simulation are summarized in Figure 49.

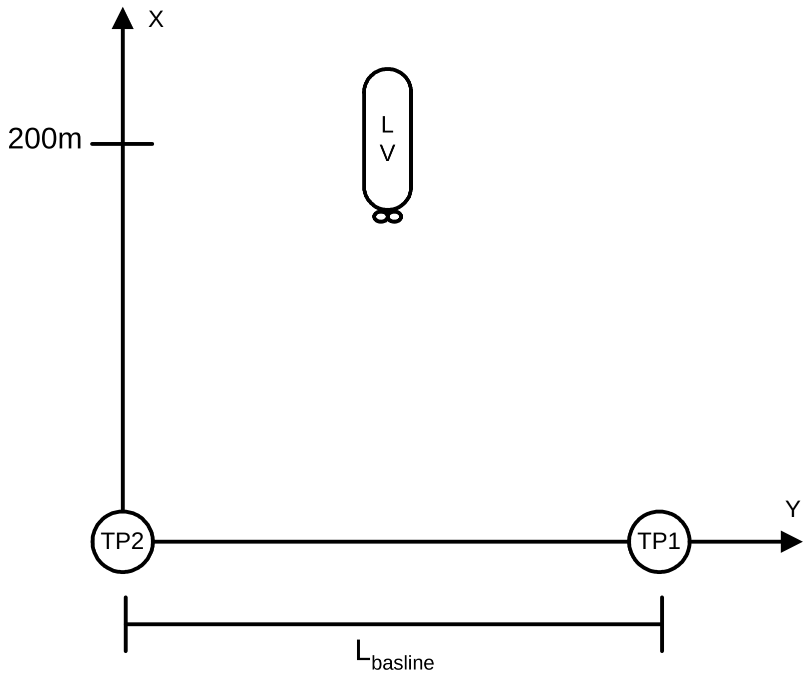


Figure 48. The test point configuration for the varying baseline length experiment where the vehicle is located 200m from the baseline and the length of the baseline is varied.

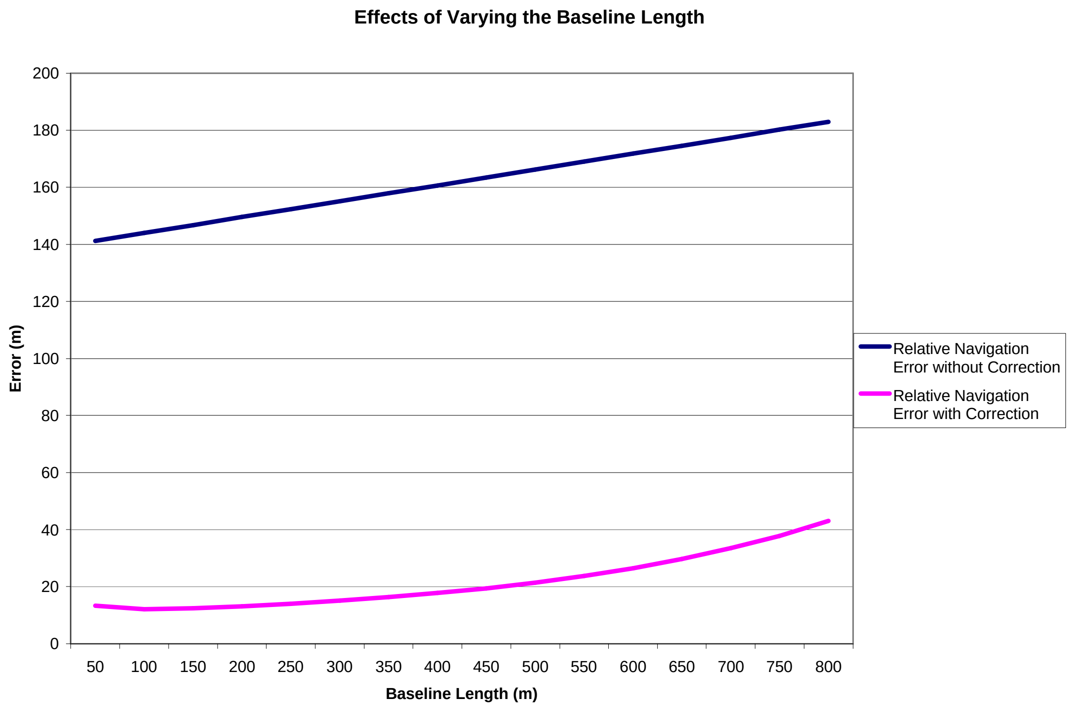


Figure 49. The results of varying the baseline length with the baseline located 200m behind the LV, DR time of 1800 seconds before the start of the correction behavior, the time to complete the correction procedure based on the distance between the test points, and the data was collected using the automatic data collection function.

Based on the simulation results the performance of the correction algorithm degrades as the baseline length increases. Considering the results presented Figure 47, a major contributing factor to the observed degradation of performance is the DR errors accumulated during the execution of the correction procedure. This occurs because as the baseline length is increased, the FV requires additional time to transit from test point 1 to test point 2, which results in additional DR errors during the execution of the relative-navigation error-correction behavior.

4. Varying Baseline Distance to the LV

The effects of varying the LV’s distance from the baseline were examined to determine its impact on the performance of the algorithm. This test was completed using the follow-the-leader test point configuration depicted in Figure 50, a LV speed of 1.5

m/s, and DR was used for 30 minutes without LBL updates before the start of the correction. The results of this simulation are summarized in Figure 51.

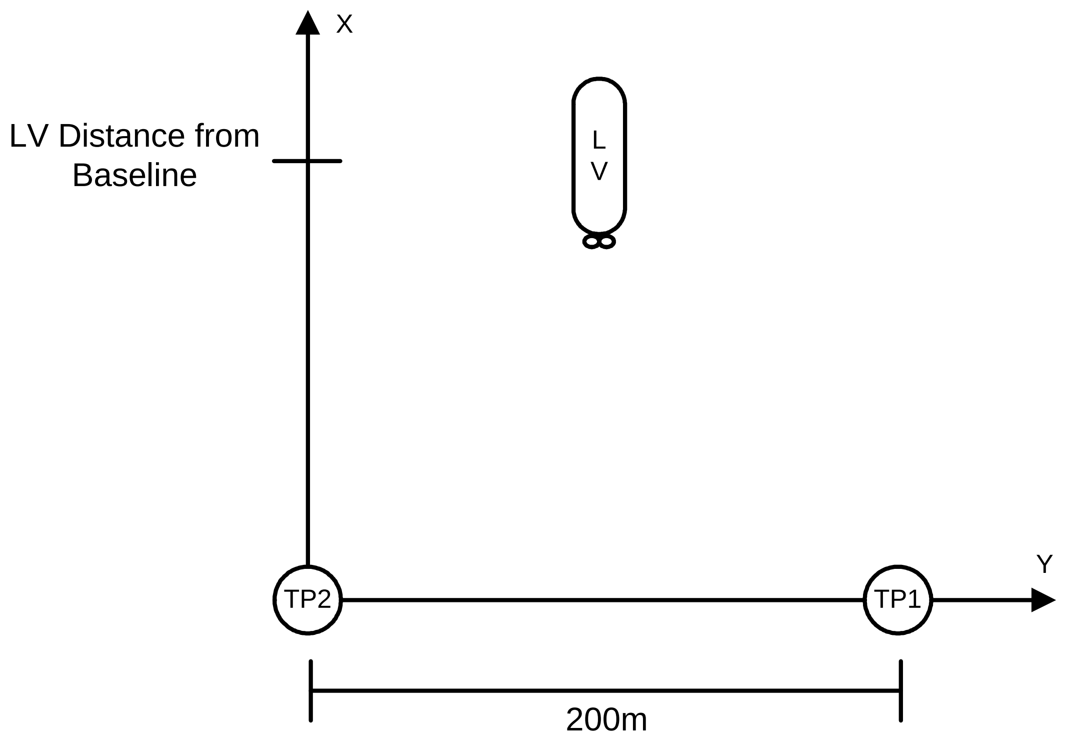


Figure 50. The test point configuration for the varying LV distance from baseline experiment where the baseline length remains fixed at 200m and the LV's distance from the baseline is varied.

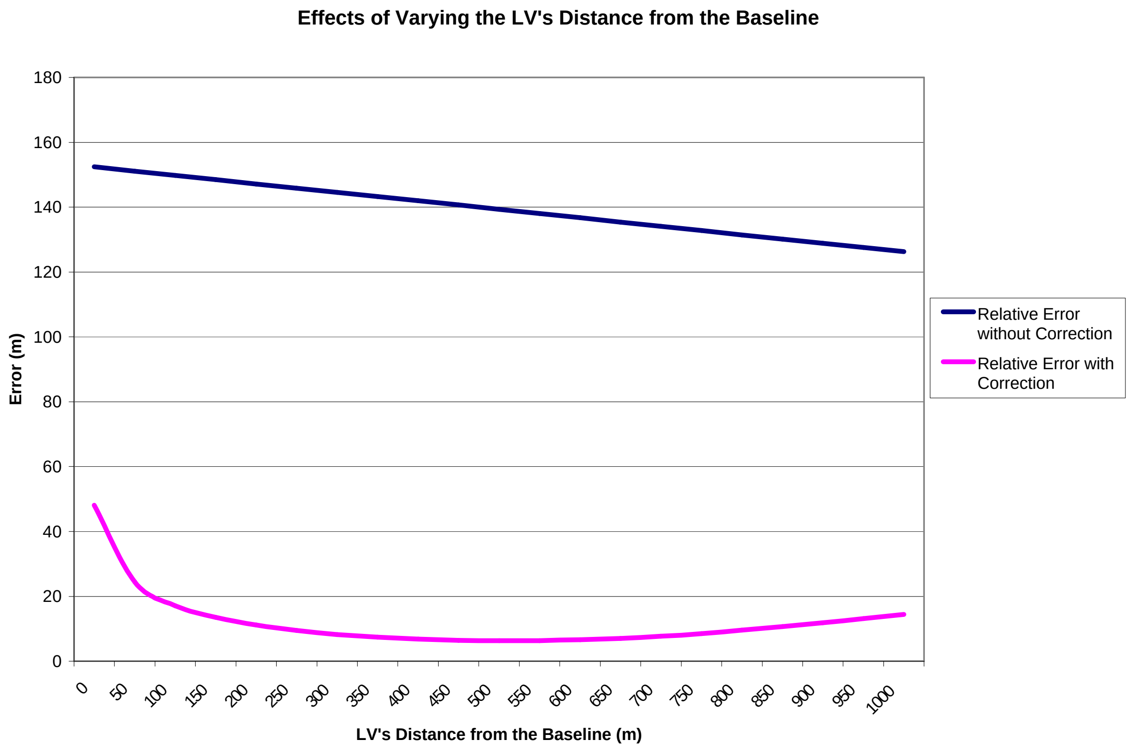


Figure 51. The results of varying the LV's distance to the baseline with a baseline length of 200m, and DR time of 1800 seconds before the correction.

Based on the simulation results, the performance of the algorithm degrades when the vehicle is close to the baseline and also begins to degrade at an approximate distance of four times the baseline length. The decreasing "relative error without correction" is an artifact of the method used for calculating the dead reckoning errors for the FV before the start of the correction procedure. The simulation assumes that both the LV and FV start at the origin (0, 0), then the LV drives on the user provided course for the amount of time indicated in the GUI's "DR Time Before Correction" field. The FV then calculates the position of test point 1 relative to the LV, and since the TPs are located behind the LV (i.e. are closer to the origin that the LV) the FV does not travel the same distance as the LV before the start of the correction. So in this experiment, as the TPs are moved further behind the LV, the FV's distance traveled before the start of the correction procedure is reduced, as is its pre-correction accumulated DR errors.

5. Effects of Random Ranging Errors

The effects of ranging errors were evaluated using the follow-the-leader test point configuration indicated in Figure 46 with a LV speed of 1.5 m/s, and DR was used without LBL updates for 30 minutes before the start of the correction. The simulation was configured such that the probability of a ranging error was 0.4 and the maximum ranging error was varied, with 49000 corrections performed for each maximum range error setting. Furthermore, the data was collected using the automatic data collection function that automatically varies the compass biases for each vehicle. The results of the simulation are presented in Figure 52.

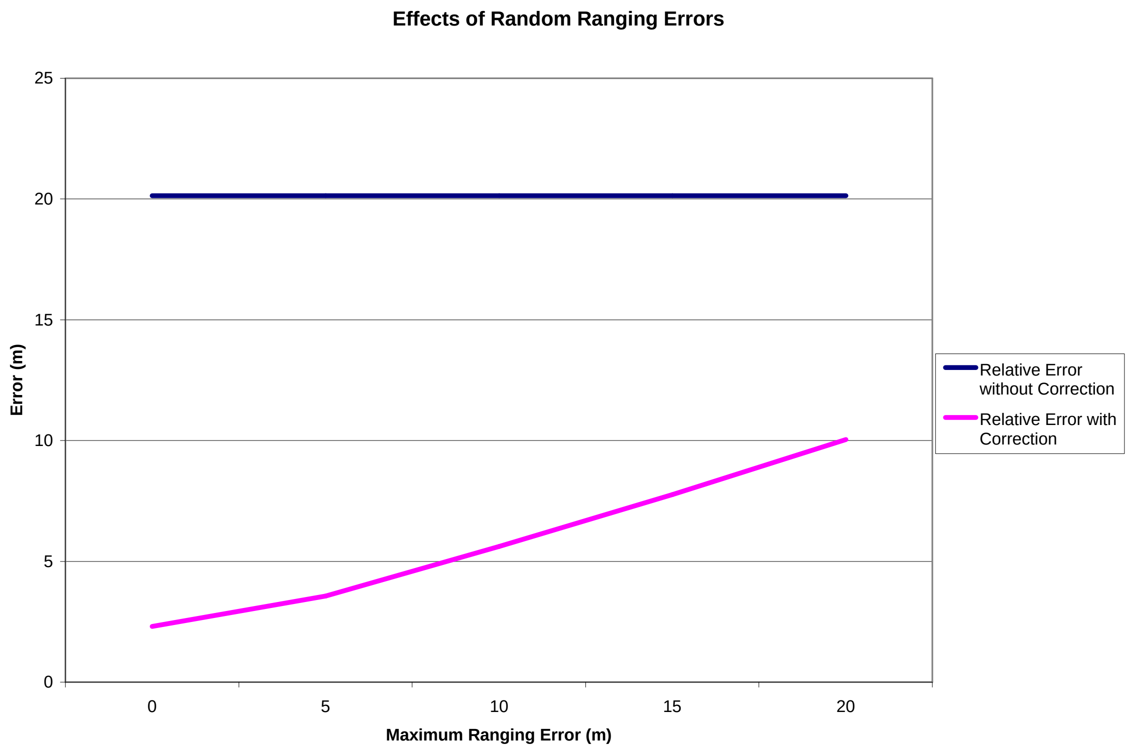


Figure 52. The impact of ranging errors on the navigation error correction algorithm's performance with a baseline length of 200m, a distance of 200m between the LV and baseline, the probability of ranging error set to 0.4, the maximum ranging error varied between 0 and 20, and compass biases determined by the automatic data collection function.

6. Single Run Results Using Optimal Settings

For this experiment a single simulation run was completed using the optimal settings determined from the previously presented simulation results. The simulation was competed using the test point configuration depicted in Figure 53, a LV compass bias of negative 3 degrees, a FV compass bias of positive 3 degrees, LV course of 90-degrees, and DR was used for 1800 seconds before the start of the correction. The results of the simulation are in Figures 53, 54, and 55.

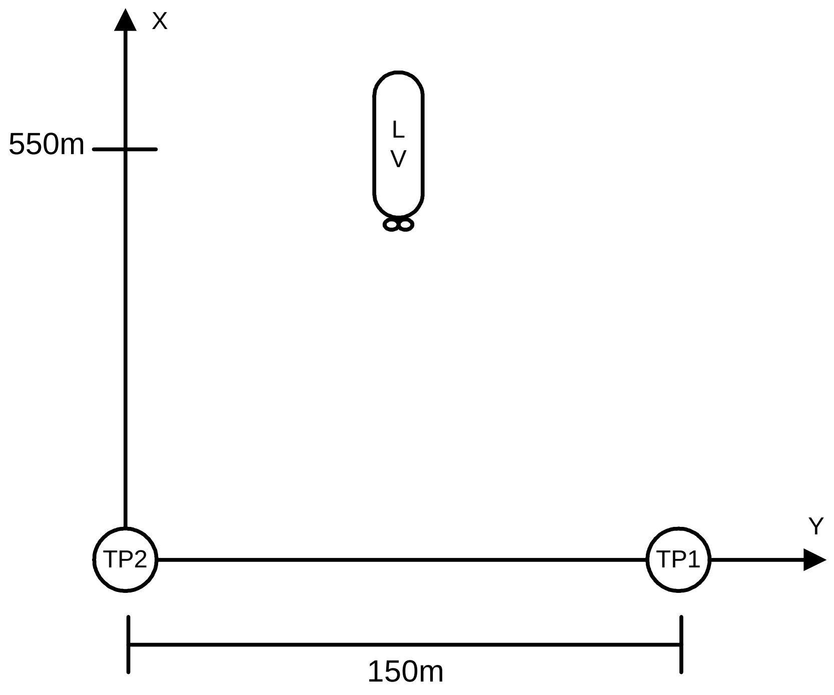


Figure 53. Test point configuration for the single run simulation using optimal settings.

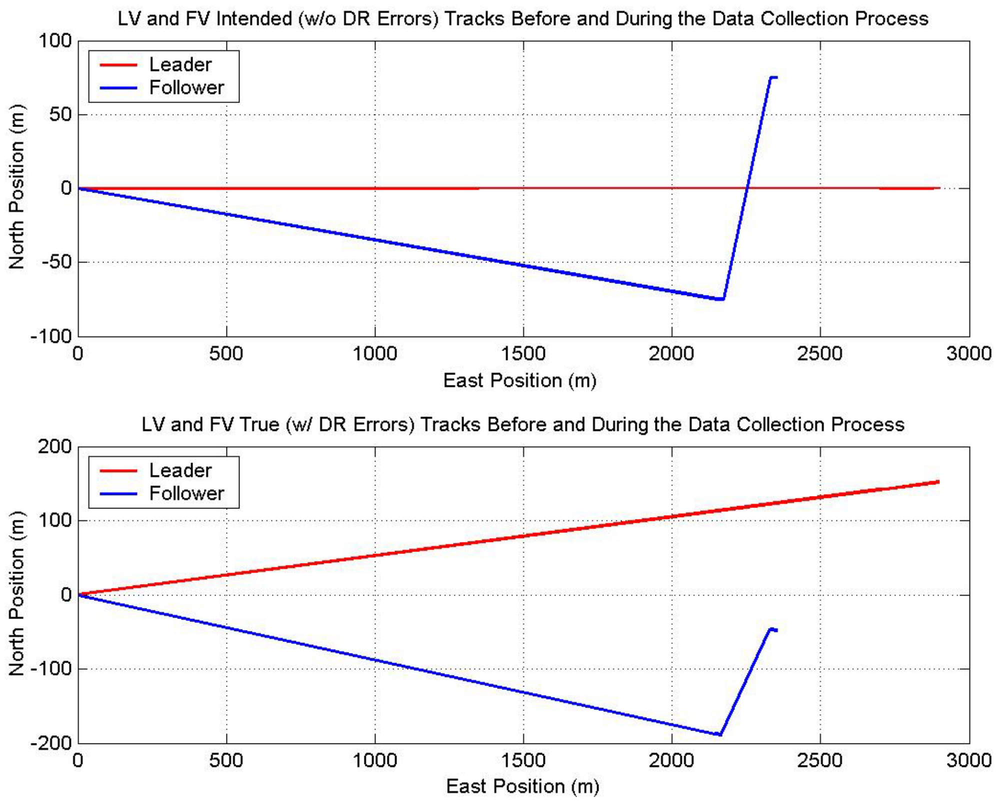


Figure 54. The true and estimated tracks for the LV and the FV before and during the correction procedure. Actual motion by the FV does not cross the LV's track as expected by DR navigation.

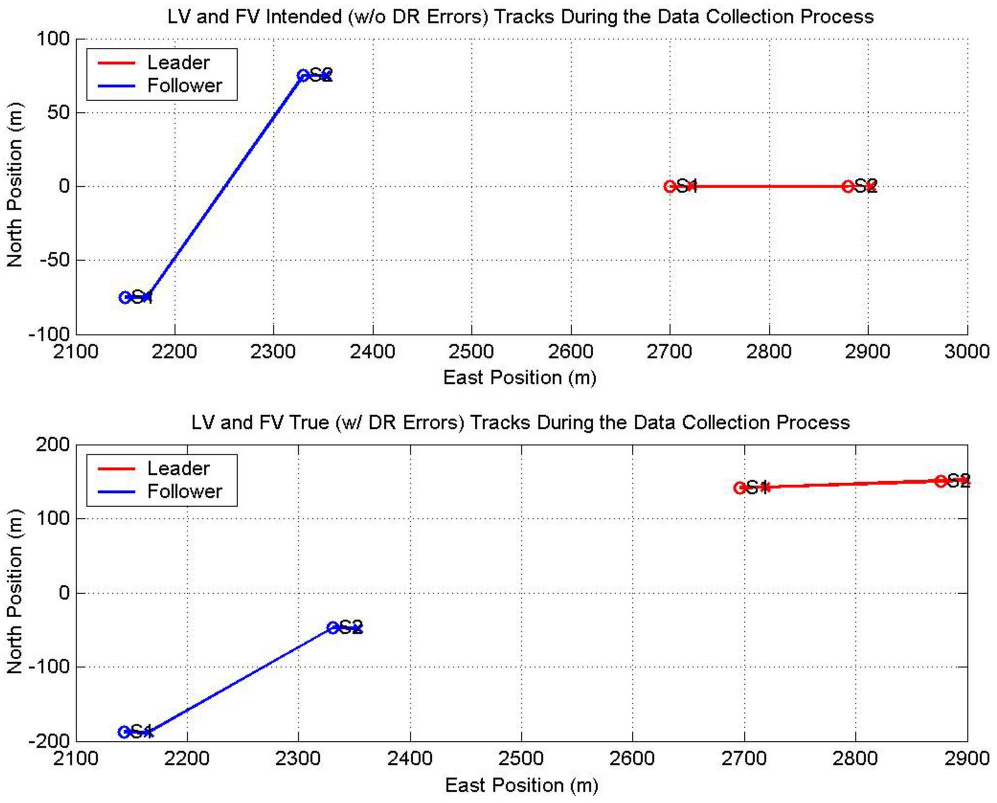


Figure 55. The LV and FV tracks during the correction procedure and the positions of the vehicles at the time each TP is reached (indicated by O). Actual motion by the FV does not cross the LV’s track as expected by DR navigation.

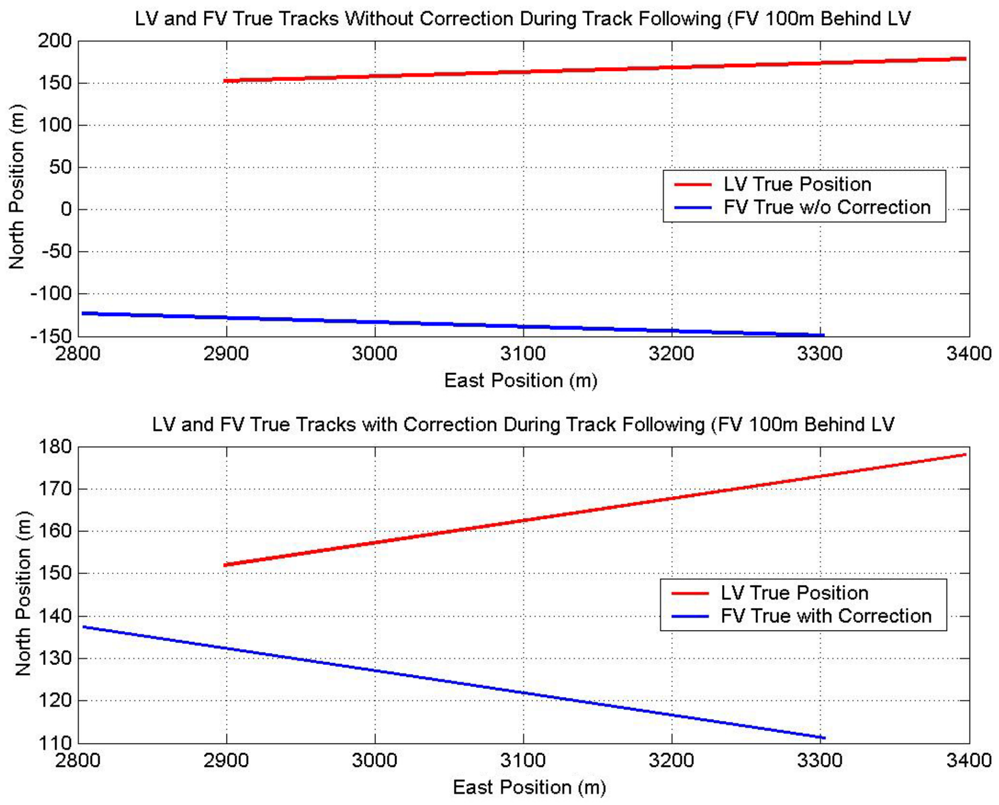


Figure 56. The results of the correction where the FV attempts to perform track following with the LV at a distance 100m behind the LV (note that DR errors continue to accumulate during the track-following behavior). The expected relative motion necessary for track following is achieved.

F. SUMMARY

The performance of the relative-navigation error-correction algorithm presented in Chapter IV was evaluated using a Matlab simulation. Based on the simulation results, the use of the proposed algorithms significantly reduces the relative errors between two vehicles. However, the algorithm is sensitive to relative-navigation errors accumulated during the execution of the correction procedure and the performance degrades when the LV is operating in close proximity to the baseline, which is a common problem with LBL systems that use two transponders.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. "FOLLOW-THE-LEADER" COOPERATIVE BEHAVIOR

A. INTRODUCTION

Cooperative behaviors enable multiple simultaneously operating vehicles to coordinate their efforts to complete a mission. This chapter presents a low-level cooperative behavior for performing follow-the-leader using AUVs, which may be built upon in the future to develop more sophisticated cooperative behaviors. The proposed solution uses a modified version of carrot or rabbit following, where the FV's waypoints follow the LV's waypoints. This chapter starts with a reexamination of the problem statement to ensure that the motivating goals are achieved.

B. PROBLEM STATEMENT

With advances in computer and sensor technologies, AUVs are now capable of performing tasks that were once thought impossible. Currently there are research efforts underway to develop new methods to improve the efficiency and effectiveness of operations completed using AUVs. One such area of research is cooperative behaviors, which enables multiple simultaneously operating AUVs to coordinate their efforts to meet the mission objectives.

A long-term goal of cooperative behaviors is for large groups of heterogeneous AUVs to operate in a single work area to complete a mission. This goal requires each vehicle to perform its operations with minimal communications because the limited bandwidth provided by ACOMMS must be shared by multiple vehicles. Furthermore, the cooperative behaviors must be robust in order to cope with the somewhat unpredictable reliability of ACOMMS.

The research presented here explores the issues associated with the development of cooperative behaviors that meet the above-stated long-term objectives of cooperative behaviors. Follow-the-leader is a low-level cooperative behavior, which provides the tools needed to develop more sophisticated cooperative behaviors that meet the long-term objectives of cooperative behaviors.

C. OBJECTIVES

The primary objective is to define the algorithm and rules needed to perform the follow-the-leader cooperative behavior using multiple AUVs. To meet the long-term goals of cooperative behaviors the following objectives were set:

1) Minimize the communications required to perform the follow-the-leader cooperative behavior.
2) Utilize the standard functionality of COTS acoustic modems, thereby eliminating the need for specialized sensors.
3) Define the follow-the-leader algorithm to enable in-transit high-speed data downloads, such as discussed earlier where ARIES follows REMUS to download data.

D. PROPOSED SOLUTION

1. Overview

This section proposes the algorithms and rules required to perform the follow-the-leader cooperative behavior.

2. Assumptions

The following assumptions were made during the development of the proposed follow-the-leader algorithm

1) The FV's control system includes a waypoint navigation capability.
2) The FV is equipped with a rendezvous algorithm, such as those currently under development by NPS' Center for AUV Research, for establishing first contact with the LV and to quickly recover after the long communications drop-outs.
3) The vehicles share a common LBL system, perform periodic GPS pop-ups, or periodically perform relative-navigation error-correction procedure presented in Chapter IV to minimize the relative navigation errors between the LV and the FV.

4) The FV can acoustically request LV status reports, which includes the LV’s current position, speed, the position of the waypoint that the LV is currently transiting to, the LV’s course to the waypoint, and the position of the next waypoint on the stack.
5) The LV sends a LV status report to the FV in the event it must alter its current waypoint as the result of obstacles, retasking, or similar events.
6) The FV is equipped with a loitering behavior, such as discussed in (Williams 2002).
7) Some depth separation is possible to reduce risk of collision during relative navigation, without excessively degrading the communications channel.

3. Approach

The proposed solution is event driven, thereby minimizing the communications requirement because information is exchanged between the LV and the FV only when necessary, which supports the long-term goals of cooperative behaviors. An alternative approach is to have the LV frequently broadcast its status information to the FV, which may minimize the effects of communications drop-outs through redundancy, but it comes at the expense of an increased bandwidth requirement. This approach differs from that of previously proposed solutions because it minimizes the bandwidth requirement, thereby making it more realistic for vehicles equipped with acoustic modems.

4. Follow-the-Leader Behavior

a. Overview

This section proposes the algorithms and rules required to perform the follow-the-leader cooperative behavior using AUVs.

b. FV Waypoint Calculation

Before the execution of the follow-the-leader behaviors it is assumed that the two vehicles are in close proximity, which may be accomplished using existing rendezvous solutions. At the start of the behavior, the FV will request a LV status report, which contains the LV’s current position, speed, position of the waypoint currently being processed (current waypoint), the LV's intended course to reach the current waypoint, and the position of the next waypoint to be processed (next waypoint). Based on the LV's current waypoint and intended course, the FV calculates its own waypoint, which is located at a position offset from the LV's waypoint, such as shown in Figure 57.

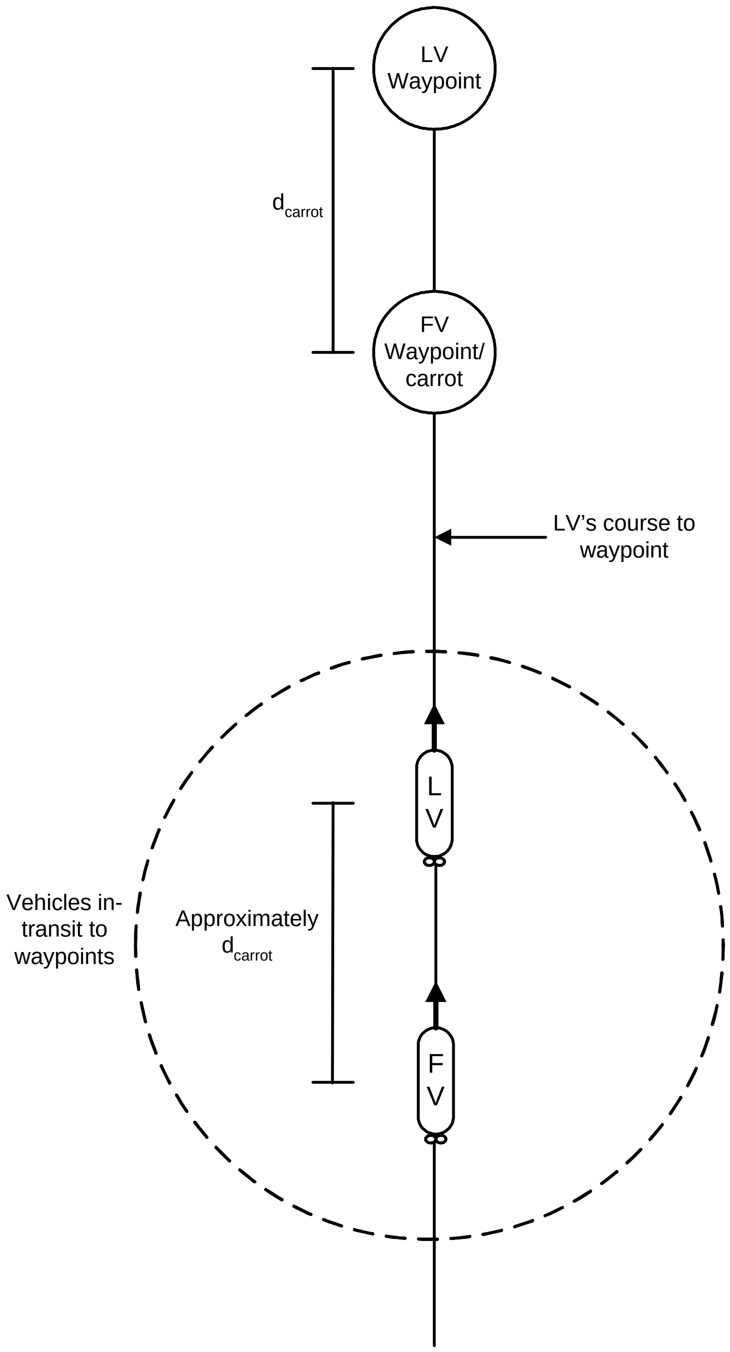


Figure 57. The position of the FV's waypoint is calculated using the position of the LV's current waypoint and its course. Both the LV and the FV share the same track in this example.

After requesting and receiving the LV’s status report, the FV must calculate the direction of the offset based on the LV’s course using Equation 26. The offset distance represents the minimal distance between the vehicles during the execution of the follow-the-leader behavior. When the LV and the FV share a common long-baseline system, the offset can remain fixed because their relative-navigation errors will likely be minimal. However, if the vehicles are using the relative-navigation error-correction procedure presented in Chapter IV, it may then be necessary to update the offset distance to reflect the relative-navigation errors accumulated since the last navigation correction, such as in Equation 27.

$$q_{carrot} = q_{LV} + 180^\circ$$

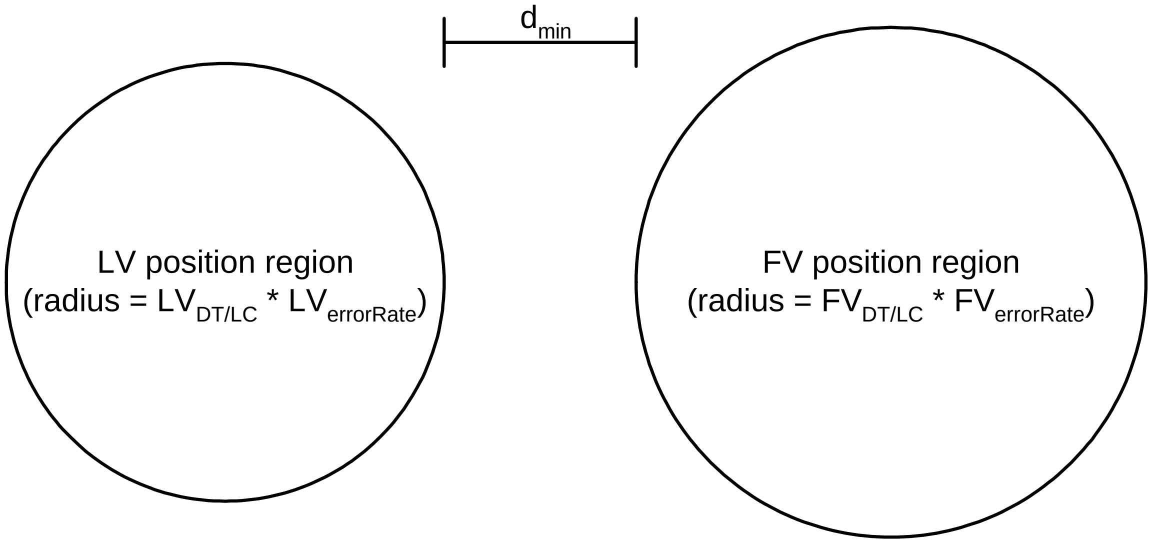
Equation 26: Direction of the offset added to the LV’s current waypoint to determine the position of the FV’s waypoint.

where,
θ_{carrot} = the angle of the carrot, which is located along the LV’s course to its waypoint
θ_{LV} = the LV’s course to its current waypoint.

$$d_{offset} = \left(LV_{DT/LC} * LV_{errorRate}\right) + \left(FV_{DT/LC} * FV_{errorRate}\right) + d_{\min}$$

Equation 27: The distance of the carrot from the LV’s waypoint when using the relative-navigation error-correction procedure.

where,
d_{offset} = the distance of the carrot from the LV’s waypoint
$LV_{DT/LC}$ = LV’s distance traveled since the last relative-navigation error-correction was performed.
$LV_{errorRate}$ = the estimated error rate for the LV. A worst case error rate, e.g. 3 %DT, may be used if the actual error rate is not known.
$FV_{DT/LC}$ = FV’s distance traveled since the last relative-navigation error-correction was performed.
$FV_{errorRate}$ = the estimated error rate for the FV. A worst case error rate, e.g. 3 %DT, may be used if the actual error rate is not known.
d_{min} = the minimum allowed distance between the vehicles.



d_{min}
LV position region
(radius = $LV_{DT/LC}$ * $LV_{errorRate}$)
FV position region
(radius = $FV_{DT/LC}$ * $FV_{errorRate}$)

Figure 58. When using Equation 27 to determine the distance of the carrot from the LV's waypoint, the worst-case minimum distance between the vehicles is d_{min} and the maximum distance is the sum of the diameters for each position regions and d_{min}.

Using the offset direction and distance, the position of the FV's waypoint is calculated using Equations 29 and 30.

$$FV_{WP/N} = LV_{WP/N} + (d_{offset} * \cos(q_{offset}))$$

Equation 29: The calculation of the north position of the FV's waypoint.

where,
$FV_{WP/N}$ = the north position of the FV's waypoint.
$LV_{WP/N}$ = the north position of the LV's waypoint.
d_{offset} = the minimum allowed distance between the vehicles
θ_{offset} = the direction of the offset relative to the position of the LV's waypoint.

$$FV_{WP/N} = LV_{WP/N} + (d_{offset} * \sin(q_{offset}))$$

Equation 30: The calculation of the east position of the FV's waypoint.

where,
$FV_{WP/E}$ = the north position of the FV's waypoint.
$LV_{WP/E}$ = the north position of the LV's waypoint.
d_{offset} = the minimum allowed distance between the vehicles
θ_{offset} = the direction of the offset relative to the position of the LV's waypoint.

c. Calculation of the LV's Transit Speed

The FV's speed is calculated such that it will reach its waypoint at approximately the same time that the LV reaches its waypoint. First the FV estimates the time required for the LV to reach its waypoint using Equation 31. Then based on this estimate, the FV uses Equation 32 to calculate its required speed.

$$LV_{timeToWP} = \frac{\sqrt{(LV_{WP/N} - LV_{Current/N})^2 + (LV_{WP/E} - LV_{Current/E})^2}}{LV_{speed}}$$

Equation 31: The estimated time required for the LV to reach its current waypoint.

where,
$LV_{timeToWP}$ = The approximate time required for the LV to reach its current waypoint.
$LV_{WP/N}$ = The north position of the LV's current waypoint.
$LV_{Current/N}$ = the LV's current north position.
$LV_{WP/E}$ = The east position of the LV's current waypoint.
$LV_{Current/E}$ = the LV's current east position.
LV_{speed} = the LV's planned speed over ground while transiting to the current waypoint.

$$FV_{Speed} = \frac{\sqrt{(FV_{WP/N} - FV_{Current/N})^2 + (FV_{WP/E} - FV_{Current/E})^2}}{LV_{timeToWP}}$$

Equation 32: The required FV speed so that the FV reaches its waypoint at approximately the same time as the LV reaches its waypoint.

where,
FV_{speed} = The transit speed of the FV.
$FV_{WP/N}$ = The north position of the FV's current waypoint.
$FV_{Current/N}$ = the FV's current north position.
$FV_{WP/E}$ = The east position of the FV's current waypoint.
$FV_{Current/E}$ = the FV's current east position.
$LV_{timeToWP}$ = The approximate time required for the LV to reach its current waypoint (from Equation 31).

d. Follow-the-Leader Behavior Execution

The proposed solution is a discrete event based system that minimizes the communications requirement by sharing data only when events occur. The follow-the-leader logic for the FV is depicted in Figure 59 and the LV's logic is in Figure 60. The FV logic includes an optional periodic check of range for vehicles using the relative-navigation error-correction algorithm presented in Chapter IV to determine when another correction may need to be performed or as an addition safeguard in noisy acoustic environments where messages are frequently lost.

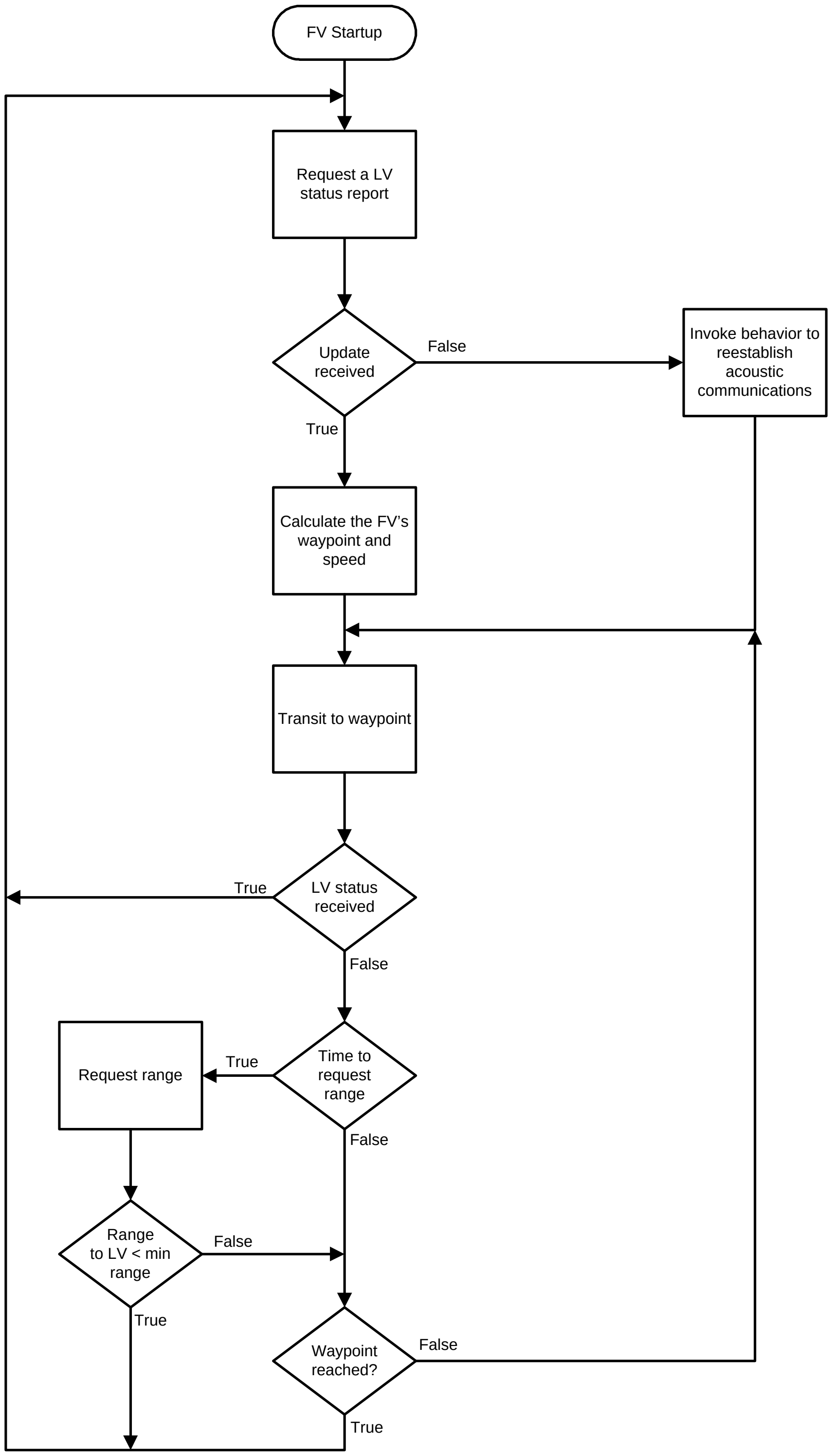


Figure 59. Flowchart of the FV's logic for the follow-the-leader behavior.

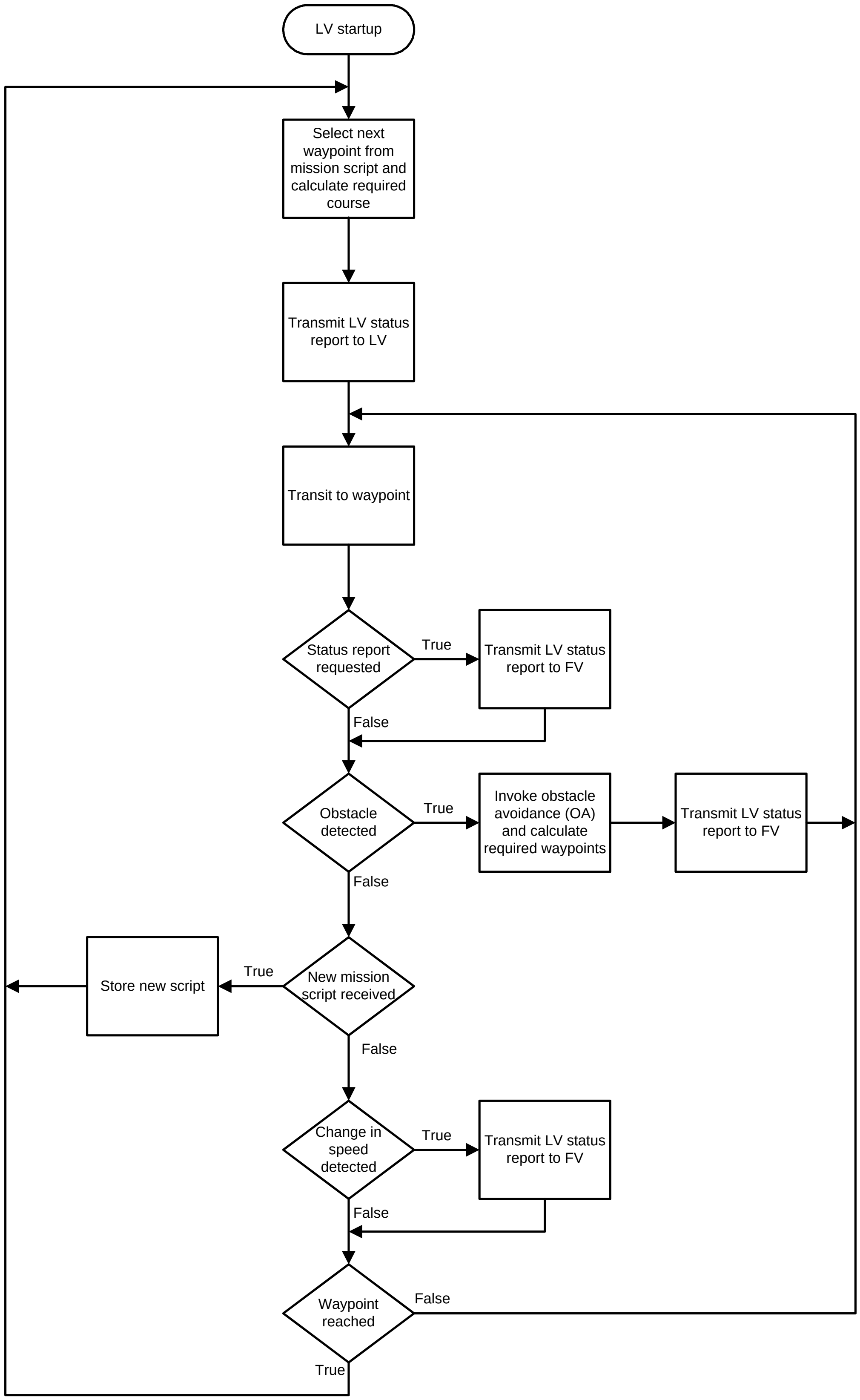

LV startup
Select next waypoint from mission script and calculate required course
Transmit LV status report to LV
Transit to waypoint
Status report requested
True
Transmit LV status report to FV
False
Obstacle detected
True
Invoke obstacle avoidance (OA) and calculate required waypoints
Transmit LV status report to FV
False
Store new script
True
New mission script received
False
Change in speed detected
True
Transmit LV status report to FV
False
Waypoint reached
False
True

Figure 60. Flowchart of the LV's logic for the follow-the-leader behavior.

There is an issue associated with the proposed solution when the LV is performing a raster search. In this situation, if the FV transits only to the calculated secondary/offset waypoint and then begins to make its turn to reach the next track, it may collide with the LV or end-up in front of the LV. To resolve this issue, after reaching the secondary waypoint the FV completes the LV's track by transiting from the offset waypoint to the LV's waypoint before executing the turn.

To add further protection against vehicle-to-vehicle collisions, it is advisable to operate the vehicles at different depths. In many cases the difference in depth will not impact the mission, such as when performing in-transit high-speed data downloads. If the vehicles are required to operate at the same depth, it is then advisable to configure the LV to automatically transmit status reports to the FV. The automatic transmission of status reports will significantly increase the bandwidth required to complete the behavior, but it will also provide necessary additional protection against vehicle-to-vehicle collisions.

E. SUMMARY

The follow-the-leader cooperative behavior enables two (or more) vehicles to follow each other, thereby enabling the performance of in-transit high-speed data downloads and other such operations. The approach proposed in this chapter is an event based system that minimizes the communications requirement by requiring vehicle-to-vehicle communications only after an event occurs, such as when the FV reaches its current waypoint and needs to know the LV's next waypoint or if the LV changes course or speed. The proposed solution uses a modified carrot-following approach, where the FV calculates a secondary waypoint that is offset from the LV's waypoint and the FV speed is set such that it reaches the secondary waypoint at approximately the same time that the LV reaches its waypoint; thereby maintaining a separation between the vehicles approximately equal to the offset between the LV's waypoint and the secondary waypoint.

VII. AUV WORKBENCH SIMULATOR

A. INTRODUCTION

The AUV Workbench is a physics-based 3D virtual world AUV simulator and mission planning tool. The workbench has been enhanced as part of this thesis to support the development and evaluation of cooperative behaviors, such as the follow-the-leader behavior presented in Chapter VI.

B. MODIFICATIONS AND IMPROVEMENTS

1. Overview

The mission planning version of the AUV workbench discussed in Chapter II has been modified to improve stability and to support the development and evaluation of cooperative behaviors.

2. Xj3D Browser Upgrade

The Xj3D browser that renders the 3D virtual world scene for the workbench has been updated to Xj3D milestone release 7 (M7). The upgrade resolves several issues experienced with previous versions of the Xj3D browser, such as stability and scene navigation issues.

3. Support for Multiple Heterogeneous Vehicles

The workbench has been modified to support the operation of multiple vehicles. Additionally, the modification allows each vehicle in the group to operate using different dynamics models and control systems, therefore supporting the operation of heterogeneous groups of AUVs.

The controller for the LV uses XML-based mission scripts to define its mission. The XML script provides the LV with a list of waypoints, which are processed in sequential order by the LV. The FV controller uses data received from the LV and the algorithms proposed in the previous chapter to dynamically calculate its waypoints. A simple loitering behavior has also been added to the FV's controller, which is a temporary behavior executed in the event acoustic communications are lost.

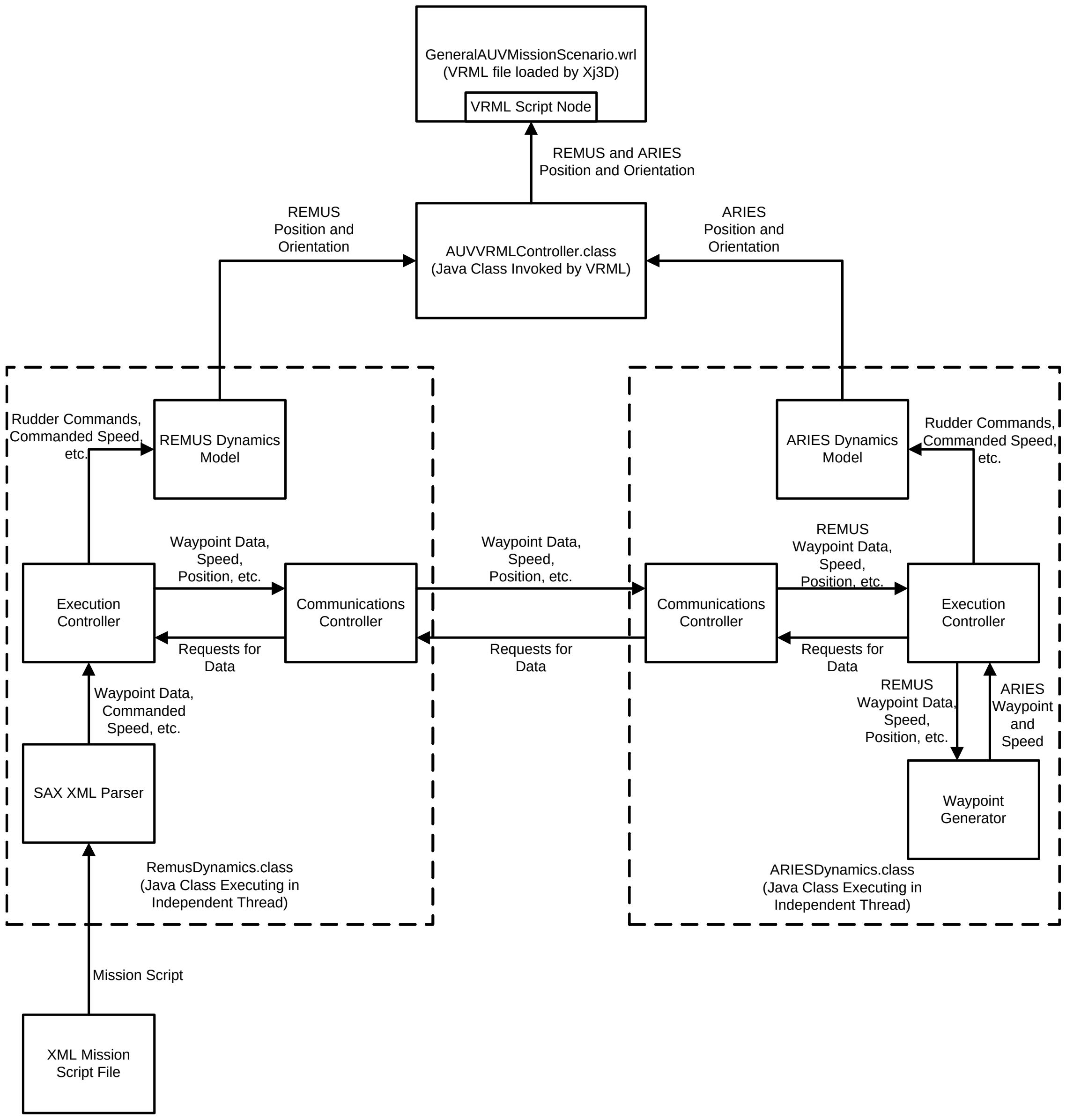

GeneralAUVMissionScenario.wrl
(VRML file loaded by Xj3D)
VRML Script Node
REMUS and ARIES Position and Orientation
REMUS Position and Orientation
AUVVRMLController.class (Java Class Invoked by VRML)
ARIES Position and Orientation
Rudder Commands, Commanded Speed, etc.
REMUS Dynamics Model
ARIES Dynamics Model
Rudder Commands, Commanded Speed, etc.
Waypoint Data, Speed, Position, etc.
Waypoint Data, Speed, Position, etc.
REMUS Waypoint Data, Speed, Position, etc.
Execution Controller
Communications Controller
Communications Controller
Execution Controller
Requests for Data
Requests for Data
Requests for Data
REMUS Waypoint Data, Speed, Position, etc.
ARIES Waypoint and Speed
Waypoint Data, Commanded Speed, etc.
SAX XML Parser
Waypoint Generator
RemusDynamics.class (Java Class Executing in Independent Thread)
ARIESDynamics.class (Java Class Executing in Independent Thread)
Mission Script
XML Mission Script File

Figure 61. Data flow diagram and module structure for the AUV Workbench with support for multiple vehicles.

4. Report Generation

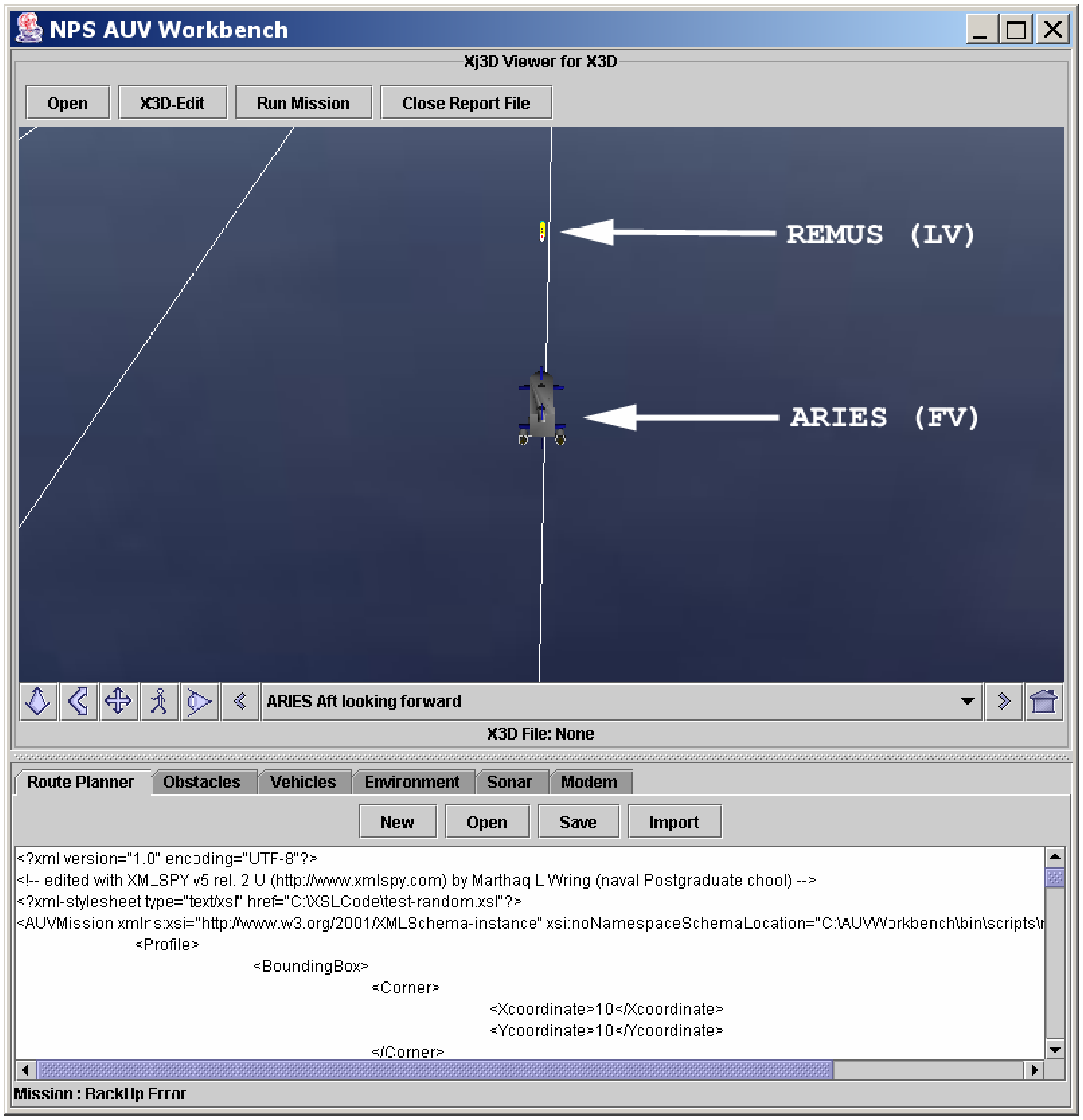
Report generation capabilities have been added to the workbench to provide a means of analyzing the performance of the behavior. During the execution of cooperative behaviors, the workbench periodically outputs the position, course, waypoint information, and status for each vehicle to a file in an ASCII comma-delimited format. The file may be opened in Matlab or Excel for post-processing and analysis to determine the performance of the cooperative behavior.

5. Vehicle-to-Vehicle Communications

Simple vehicle-to-vehicle communications capabilities have been added to the workbench to support the development of cooperative behaviors. The implementation includes the ability to force ACOMMS dead-zones by pressing a GUI button, which allows the behavior of the vehicles to be observed when vehicle-to-vehicle communications are lost. Furthermore, the status of the communications channel is included in the text output file, which enables more detailed analysis during post-processing.

C. IMPLEMENTATION OF "FOLLOW THE LEADER"

The algorithms presented in Chapter VI have been implemented in the AUV workbench to evaluate their performance. The execution of the follow-the-leader cooperative behavior without communications dropouts is depicted in Figure 62. Figure 63 depicts the behavior of the vehicles when the communications channel is lost, which causes the FV to invoke its loitering behavior while the LV continues its primary mission.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by Marthaq L Wring (naval Postgraduate chool) -->
<?xml-stylesheet type="text/xsl" href="C:\XSLCode\test-random.xsl"?>
<AUVMission xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\AUVWorkbench\bin\scripts\
        <Profile>
                <BoundingBox>
                        <Corner>
                                <Xcoordinate>10</Xcoordinate>
                                <Ycoordinate>10</Ycoordinate>
                        </Corner>
```

Mission : BackUp Error

Figure 62. The AUV Workbench user interface and 3D virtual world rendering of REMUS and ARIES performing the follow-the-leader cooperative behavior.

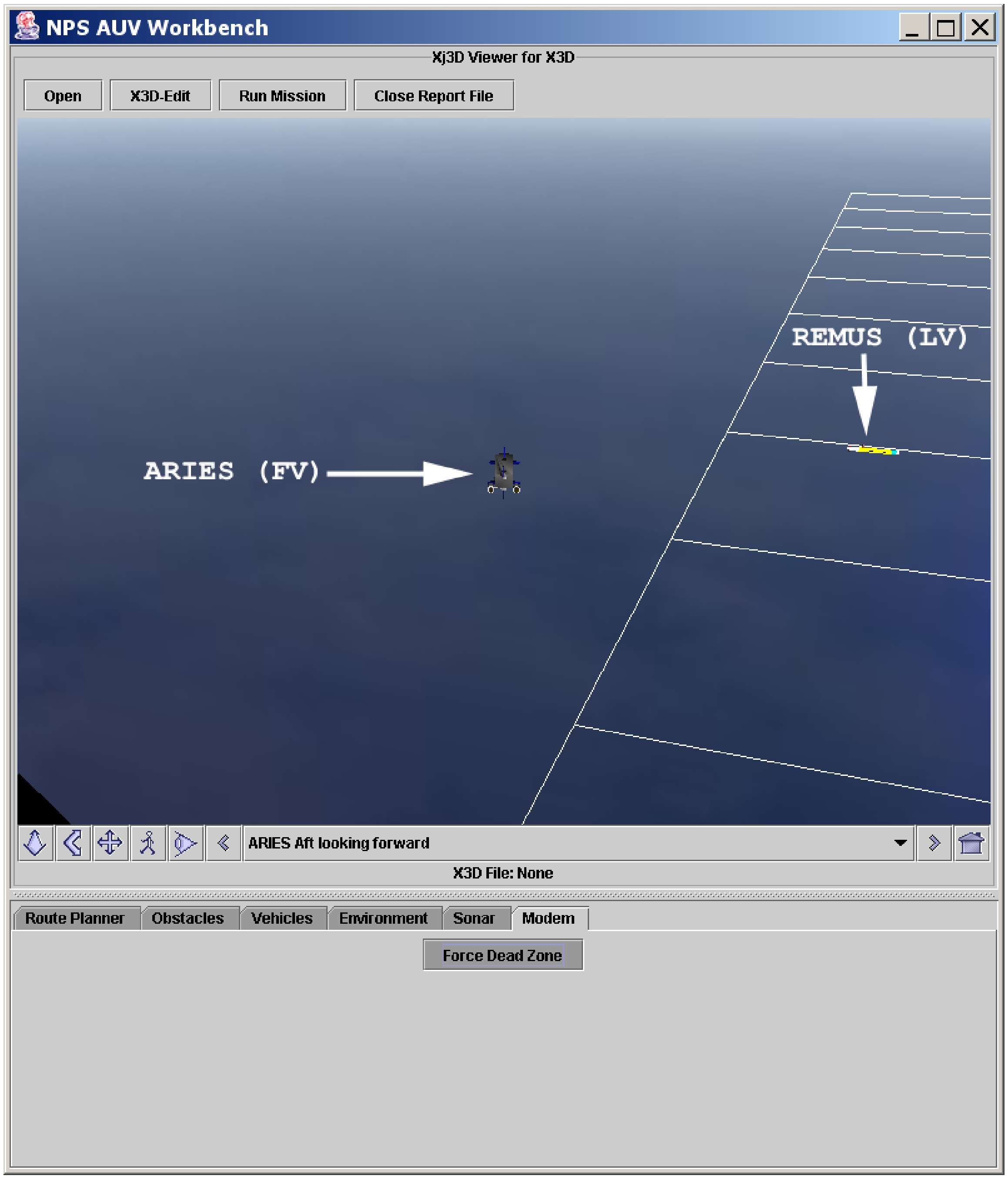


Figure 63. The AUV Workbench user interface and 3D virtual world rendering of REMUS and ARIES performing the follow-the-leader cooperative behavior when a communications drop-out occurs, which results in ARIES entering a loiter behavior while REMUS continues its raster minefield search.

D. SIMULATION RESULTS

1. Overview

This section presents the simulation results of the follow-the-leader cooperative behavior discussed in Chapter VI.

2. Performance with Perfect Communications

The performance of the follow-the-leader algorithm with perfect communications and the following settings was determined using the workbench.

1) Perfect communications.

2) The minimum distance between the LV and the FV (d_{offset}) set to 8 meters.

3) The LV speed set to 1.5 m/s

4) The FV's minimum and maximum speeds set to 1 m/s and 2 m/s respective.

5) The LV executes the 21 leg mission depicted in Figure 64.

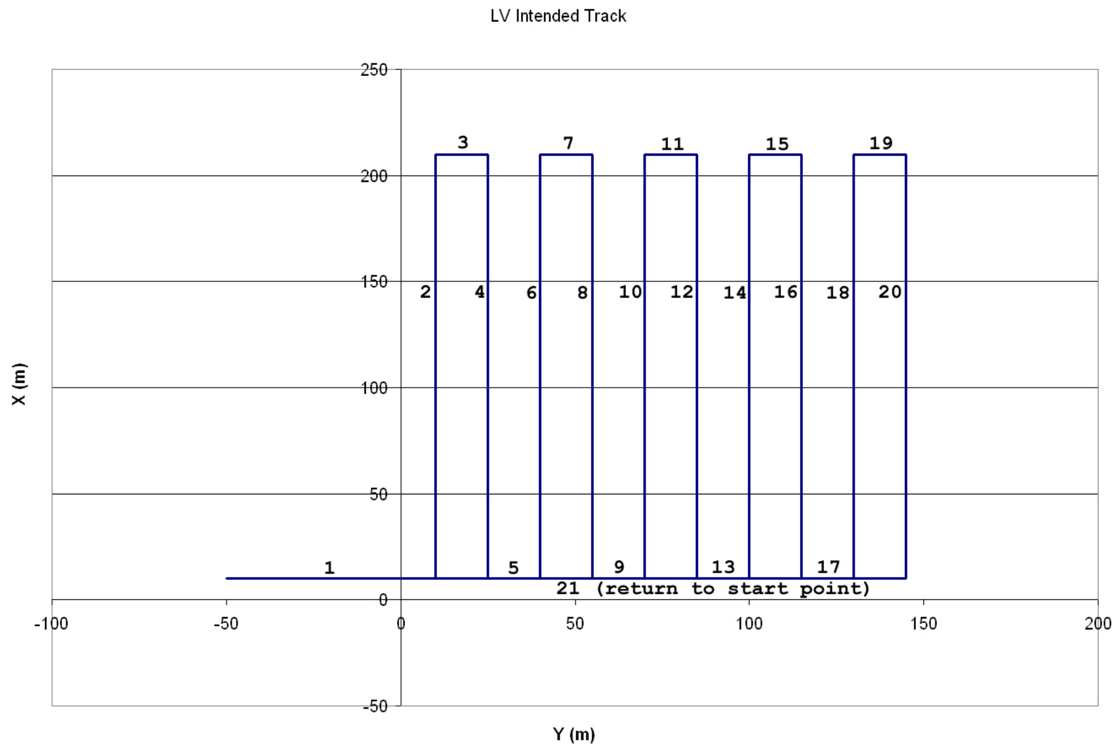


Figure 64. The intended track of the LV during test and evaluation of the follow-the-leader behavior performed using the AUV Workbench.

Under the above-stated conditions the FV maintained an average distance of 17.7m from the LV with a standard deviation of 3.8m. The minimum and maximum distances between the vehicles were 8.3m and 22.8m respectively during the mission. Figure 65 depicts the tracks for the LV and FV during the mission and Figure 66 shows the waypoints for each vehicle.

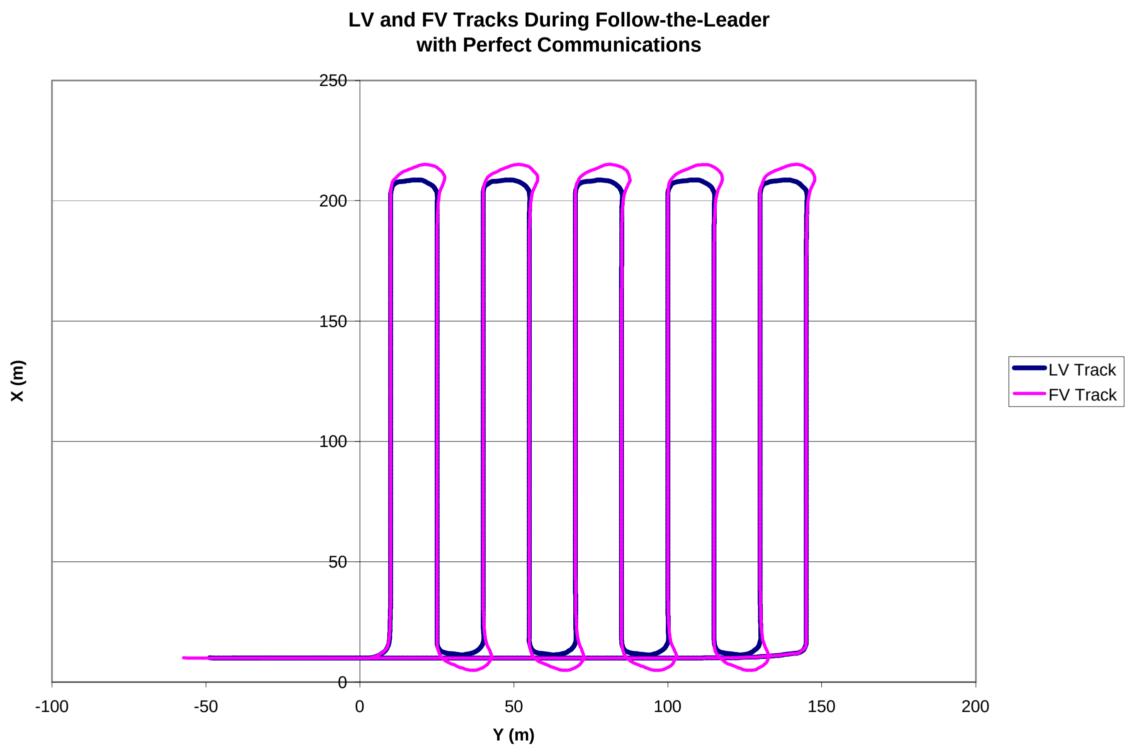


Figure 65. LV and FV tracks during the execution of the follow-the-leader cooperative behavior with perfect communications.

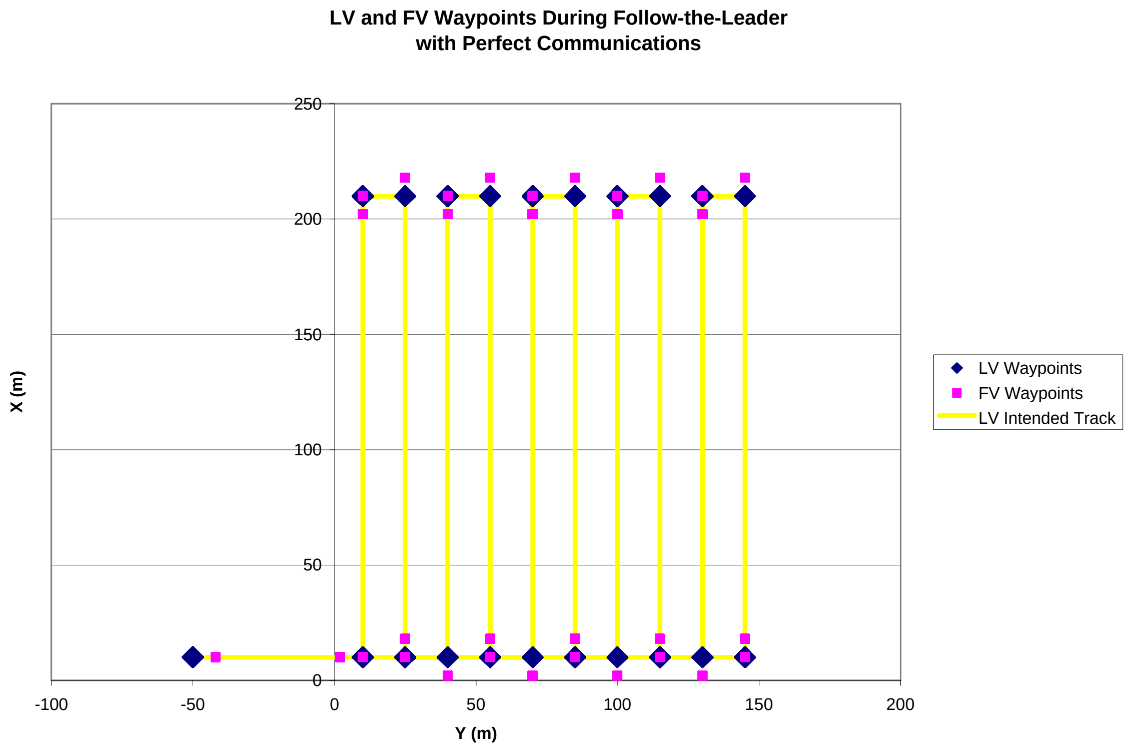


Figure 66. LV and FV waypoints during the execution of the follow-the-leader cooperative behavior with perfect communications.

3. Performance with Communications Drop-Outs

The performance of the follow-the-leader algorithm with perfect communications and the following settings was evaluated using the workbench.

1) The LV and FV enter a 60 second communications dead zone after completing the fourth leg of the 21 leg mission depicted in Figure 63.

2) The minimum distance between the LV and the FV set to 8 meters.

3) The LV speed set to 1.5 m/s

4) The FV's minimum and maximum speeds set to 1 m/s and 2 m/s respective.

Under the above conditions the FV maintained an average distance of 24m from the LV with a standard deviation of 15.5m. The minimum and maximum distances between the vehicles were 8.1m and 87.4m respectively during the mission. Figure 67 depicts the tracks for the LV and FV during the mission and Figure 68 shows the waypoints for each vehicle.

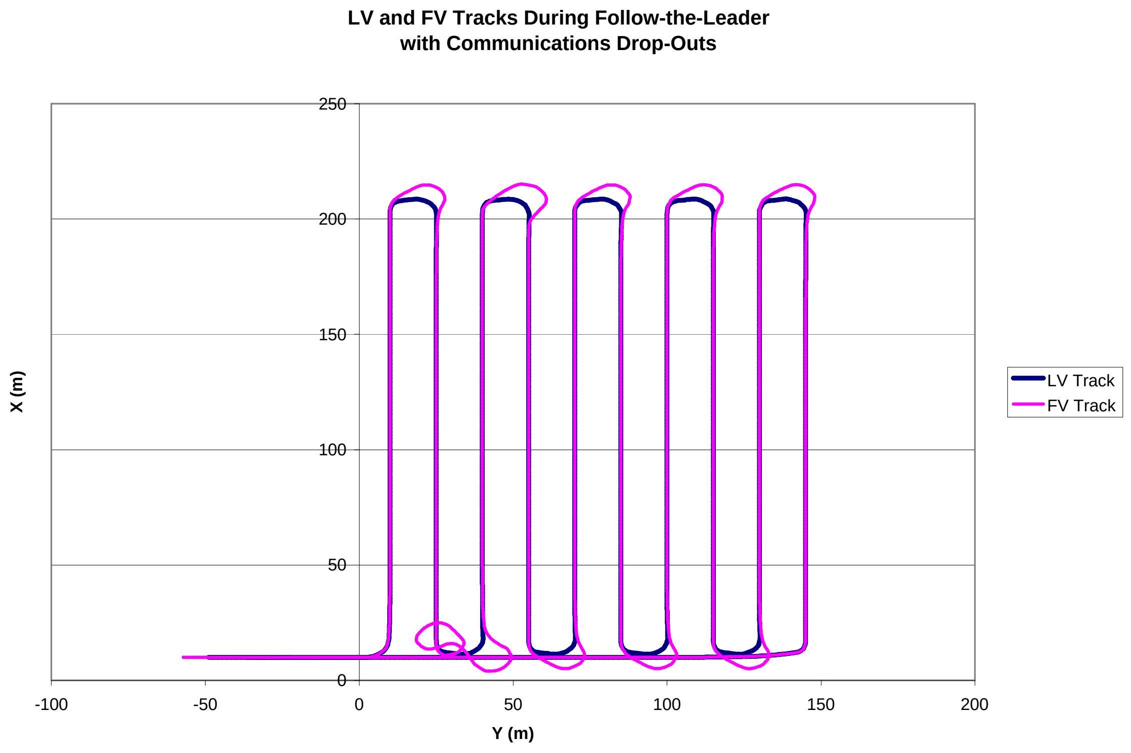


Figure 67. LV and FV tracks during the execution of the follow-the-leader cooperative behavior with a 60 second communications drop-out at the end of the fourth leg.

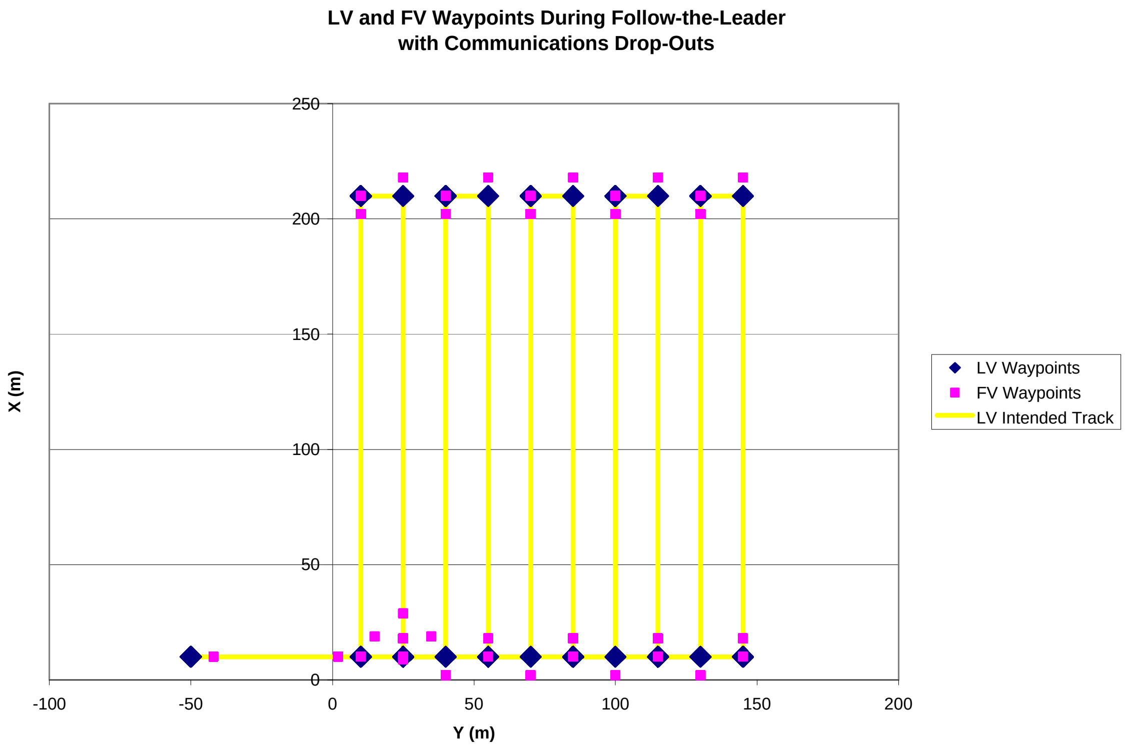


Figure 68. LV and FV waypoints during the execution of the follow-the-leader cooperative behavior with a 60 second communications drop-out at the end of the fourth leg.

E. SUMMARY

The AUV Workbench is a physics-based 3D virtual world AUV simulator and mission planning tool. The workbench has been modified to support the simulation of cooperative behaviors by providing multi-vehicle simulation capabilities, generic vehicle-to-vehicle communications, and report generation capabilities to enable performance analysis. The follow-the-leader algorithm proposed in Chapter VI was simulated using the AUV Workbench, and it operated correctly with no vehicle-to-vehicle collisions and the minimum distance between the vehicles was never less than the waypoint offset distance (d_{offset}).

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

This chapter presents the thesis research conclusions and discusses recommendations for future work.

B. GENERAL THESIS CONCLUSIONS

The research completed for this thesis provides the tools necessary to develop more sophisticated cooperative behaviors. The relative-navigation error-correction procedure enables groups of heterogeneous vehicles that do not share a common LBL system to perform cooperative behaviors. Furthermore, the enhancements to the AUV Workbench may be leveraged in the future to develop new cooperative behaviors

C. SPECIFIC CONCLUSIONS

1. Overview

This section discusses specific conclusions regarding the relative-navigation error-correction and the follow-the-leader cooperative behaviors.

2. Relative-Navigation Error-Correction Behavior

The relative-navigation error-correction behavior proposed in Chapter IV successfully minimizes the relative errors between two vehicles. The procedure operates using minimal communications bandwidth and enables the LV to perform its primary mission without disruption while the FV performs the necessary maneuvers to collect the data required to correct the relative-navigation errors. However, the proposed approach is sensitive to relative-navigation errors accumulated during the execution of the behavior and the performance begins to degrade if the LV is close to the baseline, as is common with all acoustic LBL systems that use only two transponders.

3. “Follow-the-Leader”

The proposed follow-the-leader algorithm successfully operates in the simulation environment. The distance between the LV and FV was never less than the carrot distance (d_{offset}) during the execution of a 21-leg raster search. Furthermore, in the event of ACOMMS drop-outs, the formation was quickly re-established once communications were regained.

4. AUV Workbench

The AUV Workbench now supports the simulation of cooperative behaviors executed by heterogeneous AUVs. The workbench enables the user to observe the execution of the behaviors and the report generation function enables post-processing and analyzes the results of the simulation to determine the performance of the behavior.

D. RECOMMENDED FUTURE WORK

1. Overview

This section recommends future development work for the relative-navigation error-correction procedure, the follow-the-leader cooperative behavior, and the AUV Workbench.

2. Relative-Navigation Error-Correction Procedure and Matlab Simulation

The following future work is recommended for the relative-navigation error-correction behavior:

1) Modify the Matlab simulation to include physics-based vehicle models and communications delays, dropouts, and dead-zones. Rerun the simulation to determine the impact of these factors on the performance of the proposed relative-navigation error-correction behavior.

2) Enhance the proposed algorithms to remove the test point configuration dependency, such that all calculations may be performed in the LV’s reference frame.

3) Investigate the use of multiple range readings and Kalman filtering to calculate intermediate relative-navigation error-correction factors, which progressively improve in accuracy with each additional range sample.

4) Test the proposed algorithms in-water using two AUVs.

3. Follow-the-Leader Cooperative Behavior and the AUV Workbench

The following future work is recommended for the follow-the-leader cooperative behavior:

1) Rerun the simulation of the follow-the-leader behavior with obstacles included in the field. This will require the modification of the workbench's implementation of the obstacle avoidance algorithm so that discrete waypoints are generated and transmitted to the FV.

2) Modify the workbench to include an acoustic communications model and rerun the simulation to determine the impact of communications delays and randomly lost messages.

4. AUV Workbench

The following future work is recommended for the AUV Workbench:

1) Modify the workbench to use a distributed processing approach, thereby increasing the maximum number of independent vehicles that operate in the virtual world because the workload will be distributed among multiple processors.

2) Implementation of a scripting function that allows each vehicle in the group to execute a script defining its behavior based on information obtained from other vehicles in the group. This approach facilitates the development of vehicle behaviors because the workbench code itself does not need to be modified and recompiled to test new behaviors.

3) Modification to the report generation function to include an option to generate files compatible with NPS' AUV Data Server.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ACRONYMS AND ABBREVIATIONS

Acronym / Notation	**Definition**
%DT	Percent Distance Traveled
2D	2 Dimensional
3D	3 Dimensional
ACOMMS	Acoustic Communications
ADCP	Acoustic Doppler Current Profiler
AHS	Automated Highway System
API	Application Programming Interface
ARIES	Acoustic Radio Interactive Exploratory Server
AUV	Autonomous Underwater Vehicle
CONOPS	Concept of Operations
COTS	Commercial-off-the-shelf
CTD	Conductivity, temperature, and depth (sensor)
DGPS	Differential Global Positioning System
DR	Dead Reckoning
DVL	Doppler Velocity Log (navigation sensor)
GDOP	Geometric Dilution of Precision
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LBL	Long Baseline (navigation sensor)
LOS	Line-of-sight
M&S	Modeling and Simulation
MCM	Mine Countermeasures
NPS	Naval Postgraduate School
ONR	Office of Naval Research
REMUS	Remote Environmental Measurement UnitS
RF	Radio Frequency
RIN	Reacquire, Identify, and Neutralize (MCM Operation)
SAX	Simple API for XML
SCM	Search, Classify, and Map (MCM Operation)
TPs	Test Points
UGV	Unmanned Ground Vehicle
UUV	Unmanned Underwater Vehicle
WHOI	Woods Hole Oceanographic Institute
X3D	Extensible 3D
XML	Extensible Markup Language
XSL	Extensible Style Language
XSLT	Extensible Style Language Transformation

Table 2 Acronyms and abbreviations

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB CODE FOR THE RELATIVE-NAVIGATION ERROR-CORRECTION SIMULATION

A. INTRODUCTION

This appendix provides the relative-navigation error-correction Matlab code used in Chapter V to evaluate the proposed error-correction algorithm.

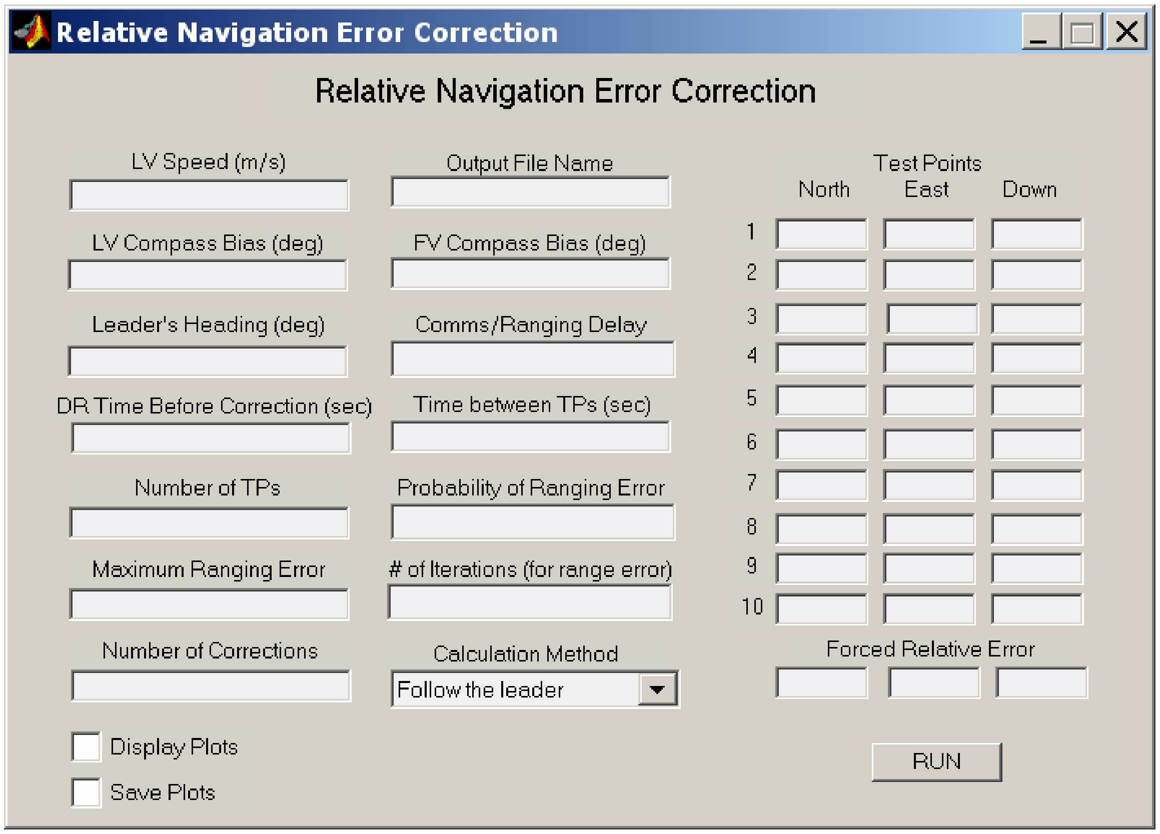
B. GRAPHICAL USER INTERFACE (GUI)

1. Overview

This section provides the handles for the Matlab GUI and the m file for processing the parameters entered by the user.

2. Graphical User Interface (GUI) Handles

GUIs created using Matlab's "guide" tool generate a set of handles to provide access to each field in the GUI. Figure 69 is the GUI for the simulation and Table 3 provides the handles for each field in the GUI.


Relative Navigation Error Correction

Relative Navigation Error Correction

LV Speed (m/s)

Output File Name

Test Points

	North	East	Down
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

LV Compass Bias (deg)

FV Compass Bias (deg)

Leader's Heading (deg)

Comms/Ranging Delay

DR Time Before Correction (sec)

Time between TPs (sec)

Number of TPs

Probability of Ranging Error

Maximum Ranging Error

# of Iterations (for range error)

Number of Corrections

Calculation Method: Follow the leader

Forced Relative Error

Display Plots

Save Plots

RUN

Figure 69. Relative-navigation error-correction Matlab graphical user interface (GUI).

GUI Field Title (from Figure 69)	**Matlab Handle Name**
LV Speed	speed
Output File Name	outputFileName
LV Compass Bias	leaderDirection
FV Compass Bias	followerDirection
Leader’s Heading	leaderHeading
Comms/Ranging Delay	commsRangingDelay
DR Time Before Correction	TimeBeforeCorrection
Time between TPs	TPTransitTime
Number of TPs	numOfTPs
Probability of Ranging Error	pRangingError
Maximum Ranging Error	maxRangingError
# of Iterations (for range error)	numberOfIterations
Number of Corrections	numberOfCorrections
Calculation Method	calculationMethod
Test Point x North (where x is the TP number)	TPxN (where x is the TP number)
Test Point x East (where x is the TP number)	TPxE (where x is the TP number)
Test Point x Down (where x is the TP number)	TPxD (where x is the TP number)
Forced Relative Error North	forcedErrorN
Forced Relative Error East	forcedErrorE
Forced Relative Error Down	forcedErrorD
Display Plots	displayPlots
Save Plots	savePlots

Table 3 The graphical user interface (GUI) handles for the Matlab relative-navigation error-correction computations.

C. GUI DATA PROCESSING

1. Overview

This section presents the code for processing the data entered in the Matlab GUI depicted in Figure 68.

2. DataEntryGUI.m

```
function varargout = dataentrygui(varargin)
% This function is the interface to the user input GUI.
% DATAENTRYGUI Application M-file for dataentrygui.fig
%    FIG = DATAENTRYGUI launch dataentrygui GUI.
%    DATAENTRYGUI('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 15-Sep-2003 17:31:23

if nargin == 0  % LAUNCH GUI

global figureNumber;    %setup the figure number

figureNumber = 1;       %placed here so init only once

    fig = openfig(mfilename,'reuse');

       % Use system color scheme for figure:
       set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

       % Generate a structure of handles to pass to callbacks, and store it.
       handles = guihandles(fig);
       guidata(fig, handles);

       if nargout > 0
              varargout{1} = fig;
       end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

       try
              [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
       catch
              disp(lasterr);
       end

end


% Callback function for processing the run button.
function varargout = runbutton_Callback(h, eventdata, handles, varargin)
global numberOfIterations;
global leaderDRErrorDir;
global followerDRErrorDir;

collectGUIData(handles);    %collect the GUI data

if(leaderDRErrorDir == -1 & followerDRErrorDir == -1)
    autoFlag = 1;
else
    autoFlag = 0;
end
```

```
for(k=1:numberOfIterations)
    CorrectNavErrors;  %call the primary processing function
    if(autoFlag ==1)
        leaderDRErrorDir = -1;
        followerDRErrorDir = -1;
    end
end

disp('DONE');


% ********************************************************************
%            START OF GUI INTERFACE FUNCTIONS
% ********************************************************************

% Collects the data from the GUI and places the information in
% global variables for use by other functions.
%
function collectGUIData(handles)
global GUIHandles;
GUIHandles = handles;

global numberOfCorrections;
numberOfCorrections = getNumberOfCorrections;
    %the number of correction attempts to complete


global calculationMethod;
calculationMethod = getCalcMethod;
    %the calculation method to be used to determine the correction
    %factor
    % =1 for simple trig method
    % =2 for the processor intensive over-determined system of
    % non-linear distance equations

global numberOfIterations;
    numberOfIterations = getNumberOfIterations;
    %the number of times to run the sim
    %intended for use with the probablity of ranging error

global speed;
speed = getSpeed;
    %speed of the vehicles before and during calibration (m/s)

global outputFile;
outputFile = getFileName;
    %the name of the file to store the simulation data

global leaderDRErrorDir;
leaderDRErrorDir = getLVDRErrorDir;
    %the direction of the lead vehicle's dead reckoning errors (degrees)

global followerDRErrorDir;
followerDRErrorDir = getFVDRErrorDir;
    %the direction of the follower vehicle's dead reckoning errors (degrees)


global leaderHeading;
leaderHeading = getHeading;
    %the heading of the leader during the calibration (degrees)

global commsRangeDelay;
commsRangeDelay = getCommsRangeDelay;
```

```
    %the delay between when the vehicle-to-vehicle comms is received and
    %when the ranging information is received (sec)

global preCorrectionDRTime;
preCorrectionDRTime = getPreCorrectionDRTime;
    %the length of time the vehicles are dead reckoning before the start
    %of the calibration (sec).  This is used to determine the relative
    %navigation errors between the LV and the FV


global NumOfTPs;
NumOfTPs = getNumOfTPs;
    %the number of test points to use for the calibration

global PRangingError;
PRangingError = getPRangingError;
    %probability of a spurious ranging error (other than white noise)

global maxRangingError;
maxRangingError = getMaxRangingError;
    %the maximum error associated with spurious ranging values

global testPoints;
testPoints = getTestPoints;
    %the test points to be used for the calibration.  These are the positions
    %of the FV relative to the LV's position.  The format is (north,east,down)
    %so if these points are to be plotted where y/up is north remember to
    %adjust the ordering.

global TPTransitTime;
TPTransitTime = getTPTransitTime;
    %the time required for the FV to transit between the each test point (sec)

global savePlots;
savePlots = getSavePlots;
    %flag indicating if the plots should be saved

global displayPlots;
displayPlots = getDisplayPlots;

global forcedError;
forcedError = getForcedError;

% Gets the components of the error to be forced
% as the initial relative navigation error.
function forcedError = getForcedError
global GUIHandles;
N = str2double(get(GUIHandles.forcedErrorN, 'String'));
E = str2double(get(GUIHandles.forcedErrorE, 'String'));
D = str2double(get(GUIHandles.forcedErrorD, 'String'));
if(isnan(N) | isnan(E) | isnan(D))
    forcedError = [0, 0, 0];
else
    forcedError = [N, E, D];
end


% Gets the value of the GUI check box indicating
% if the plots should be displayed.
function displayPlots = getDisplayPlots
global GUIHandles;
if(get(GUIHandles.displayPlots,'Value')  ==  get(GUIHandles.displayPlots,'Max'))
%if checked
```

```
    displayPlots = 1;
else
    displayPlots = 0;
end


% Gets the value of the GUI check box indicating
% if the plots should be saved.
function savePlots = getSavePlots
global GUIHandles;
if(get(GUIHandles.savePlots,'Value') == get(GUIHandles.savePlots,'Max'))     %if
checked
    savePlots = 1;
else
    savePlots = 0;
end


% Gets the test points from the GUI.  These points
% are based on the leader's local reference frame (N,E,D)
% where the origin is the LV's current position
%              _________
%           8<_________> North->
%
% Returns a 3xN where N is the number of test points
% indicated by getNumOfTPs or E-1 where E is the first
% row containing a NAN value.  N will have a minimum
% value of 3 and if a NAN occurs in a row < 3 the row
% will contain all zeros.
function testPoints = getTestPoints
global GUIHandles
%Test Point 1
X = str2double(get(GUIHandles.TP1N, 'String'));
Y = str2double(get(GUIHandles.TP1E, 'String'));
Z = str2double(get(GUIHandles.TP1D, 'String'));
if(isnan(X) | isnan(Y) | isnan(Z))
    testPoints = [0, 0, 0];
else
    testPoints = [X, Y, Z];
end
% Test point 2
X = str2double(get(GUIHandles.TP2N, 'String'));
Y = str2double(get(GUIHandles.TP2E, 'String'));
Z = str2double(get(GUIHandles.TP2D, 'String'));
if(isnan(X) | isnan(Y) | isnan(Z))
    testPoints = [testPoints; 0, 0, 0];
else
    testPoints = [testPoints; X, Y, Z];
end


%test point =3
if(getNumOfTPs >2)
    % Test point 3
    X = str2double(get(GUIHandles.TP3N, 'String'));
    Y = str2double(get(GUIHandles.TP3E, 'String'));
    Z = str2double(get(GUIHandles.TP3D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end;
```

```
%test point 4
if(getNumOfTPs >3)
    X = str2double(get(GUIHandles.TP4N, 'String'));
    Y = str2double(get(GUIHandles.TP4E, 'String'));
    Z = str2double(get(GUIHandles.TP4D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 5
if(getNumOfTPs >4)
    X = str2double(get(GUIHandles.TP5N, 'String'));
    Y = str2double(get(GUIHandles.TP5E, 'String'));
    Z = str2double(get(GUIHandles.TP5D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 6
if(getNumOfTPs >5)
    X = str2double(get(GUIHandles.TP6N, 'String'));
    Y = str2double(get(GUIHandles.TP6E, 'String'));
    Z = str2double(get(GUIHandles.TP6D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 7
if(getNumOfTPs >6)
    X = str2double(get(GUIHandles.TP7N, 'String'));
    Y = str2double(get(GUIHandles.TP7E, 'String'));
    Z = str2double(get(GUIHandles.TP7D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 8
if(getNumOfTPs >7)
    X = str2double(get(GUIHandles.TP8N, 'String'));
    Y = str2double(get(GUIHandles.TP8E, 'String'));
    Z = str2double(get(GUIHandles.TP8D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 9
if(getNumOfTPs >8)
```

```
    X = str2double(get(GUIHandles.TP9N, 'String'));
    Y = str2double(get(GUIHandles.TP9E, 'String'));
    Z = str2double(get(GUIHandles.TP9D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end

%test point 10
if(getNumOfTPs >9)
    X = str2double(get(GUIHandles.TP10N, 'String'));
    Y = str2double(get(GUIHandles.TP10E, 'String'));
    Z = str2double(get(GUIHandles.TP10D, 'String'));
    if(isnan(X) | isnan(Y) | isnan(Z))
        testPoints = [testPoints; 0, 0, 0];
    else
        testPoints = [testPoints; X, Y, Z];
    end
end


% Gets the calculation method to be used for
% generating the correction factor.
% Options are
% simple trig - uses the least
% processor intensive method for calculating
% the correction factor
% system of eqns - uses the most processor
% intensive method for calculating the
% correction factor (over-determined non-linear
% system of distance equations)
% returns 1 for follow the leader
% returns 2 for system of equations
% returns 3 for leap frog
function calcMethod = getCalcMethod
global GUIHandles;
calcMethod = get(GUIHandles.calculationMethod, 'Value');


% Gets the max error for spurious ranging values
% Returns 0 if the user's input is invalid
function maxError = getMaxRangingError
global GUIHandles;
maxError = str2double(get(GUIHandles.maxRangingError, 'String'));
if(maxError < 0 | isnan(maxError))
    set(GUIHandles.maxRangingError, 'String', '0');
    maxError = 0;
end


% Gets number of corrections to be performed.
% Returns 1 if the user's input is invalid
function iterations = getNumberOfCorrections
global GUIHandles;
iterations = str2double(get(GUIHandles.numberOfCorrections, 'String'));
if(iterations <= 0 | isnan(iterations))
    set(GUIHandles.numberOfCorrections, 'String', '1');
    iterations = 1;
end
```

```
% Gets the number of iterations to be executed
function numberOfIterations = getNumberOfIterations
global GUIHandles;
numberOfIterations = str2double(get(GUIHandles.numberOfIterations, 'String'));
if(numberOfIterations < 1 | isnan(numberOfIterations))
    set(GUIHandles.numberOfIterations, 'String', '1');
    numberOfIterations = 1;
end



% Gets the probability of receiving a spurious range value
% returns 0 if an invalid value is entered
function prob = getPRangingError
global GUIHandles;
prob = str2double(get(GUIHandles.pRangingError, 'String'));
if(prob < 0 | prob > 1 | isnan(prob))
    prob = 0;
    set(GUIHandles.pRangingError, 'String', num2str(prob));
end


% Gets the number of test points used to perform the calibration
% from the GUI
% Valid value for simple trig calc method is 2
%                 system of eqns is 3-10
% note that getCalcMethod must be called first so the calc
% method is available through calculationMethod.
function count = getNumOfTPs
global GUIHandles;
global calculationMethod;
count = str2double(get(GUIHandles.numOfTPs, 'String'));
if(calculationMethod == 1 | calculationMethod == 3)
    if(count ~= 2 | isnan(count))
        set(GUIHandles.numOfTPs, 'String', '2');
        count = 2;
    end
else
    if(count < 3 | count > 10 | isnan(count))
        set(GUIHandles.numOfTPs, 'String', '3');
        count = 3;
    end
end


% Gets the transit time between test points from the GUI
% Returns 0 if the user's input is invalid
function time = getTPTransitTime
global GUIHandles;
global calculationMethod;
global testPoints;
global speed;
time = str2double(get(GUIHandles.TPTransitTime, 'String'));
if(time < 0 | isnan(time))
    if(calculationMethod == 1 | calculationMethod == 3)
        dist = sqrt((testPoints(1,1) - testPoints(2,1))^2 + (testPoints(1,2) -
testPoints(2,2))^2);
            %get the distance between the test points
        time = (dist/speed)*2;

        dist = (abs(testPoints(1,1) - testPoints(2,1))) + (abs(testPoints(1,2)
- testPoints(2,2)));
```

```
        %calculation of worst case distance if the FV
        %takes the least efficient route and the FV's
        %speed is 2x the LV's speed
        % The worst case route
        % TP2
        %  |
        %  |
        %  |
        %  |
        %  |
        %  --------------------------TP1
        %
    time = (dist/speed)*1.05;
        %Time required to transit worst case path if the
        %FV's speed is 2x the FV's speed.  An additional
        %5% time is added to account for turns to ensure
        %the worst case is represented.
    set(GUIHandles.TPTransitTime, 'String', num2str(time));
    else
        set(GUIHandles.TPTransitTime, 'String', '0');
        time = 0;
    end
end



% Gets the length of time the vehicles used dead reckoning
% before starting the correction.  This value is used to
% calculate the accumulated DR errors.
% Returns 0 if the user's input is invalid
function time = getPreCorrectionDRTime
global GUIHandles;
time = str2double(get(GUIHandles.DRTimePreCorrection, 'String'));
if(time < 0 | isnan(time))
    set(GUIHandles.DRTimePreCorrection, 'String', '0');
    time = 0;
end


% Gets the elapsed time between when the LV comms and ranging are received.
% Returns 0 if the user's input is invalid
function time = getCommsRangeDelay
global GUIHandles;
time = str2double(get(GUIHandles.commsRangingDelay, 'String'));
if(time < 0 | isnan(time))
    set(GUIHandles.commsRangingDelay, 'String', '0');
    time = 0;
end


% Gets the leader's speed from the user interface and converts it to double.
% Returns 0 if the user's input is invalid
function speed = getSpeed
global GUIHandles;
speed = str2double(get(GUIHandles.speed, 'String'));
if(speed < 0 | isnan(speed))
    set(GUIHandles.speed, 'String', '0');
    speed = 0;
end


% Gets the heading from the user interface and converts it to double.
% Returns 0 if the user's input is invalid
```

```
function heading = getHeading
global GUIHandles;
heading = str2double(get(GUIHandles.leaderHeading, 'String'));
if(heading < 0 | heading>= 360 | isnan(heading))
    set(GUIHandles.leaderHeading, 'String', '0');
    heading = 0;
end


% Gets the output file name from the user interface,
% appends a .txt extension to the file name, appends
% a data\ path to the file name, and
% creates a directory for the figures
%
function fileName = getFileName
global GUIHandles;
global figDirectory; %Dir to hold generated figures

if(exist('data') ~= 7)  %if data subdirectory doesn't exist
    [status,msg] = mkdir('data');
    if(status ~=1)
        disp('Error creating \data directory');
    end;
end;

fileName = get(GUIHandles.outputFileName, 'String');
if(isempty(fileName))
    fileName = 'data\1.txt'
    count = 1;
    while(exist(fileName) ~= 0)
        count = count+1;
        fileName = strcat('data\', num2str(count), '.txt');
    end;
    figDirectory = strcat('data\', num2str(count));
    set(GUIHandles.outputFileName, 'String', num2str(count));
else
    figDirectory = strcat('data\', fileName);
    fileName = strcat('data\',fileName, '.txt');
end;


if(exist(figDirectory) ~= 7)  %if figure subdirectory doesn't exist
    [status,msg] = mkdir(figDirectory);
    if(status ~=1)
        disp('Error creating figure directory');
    end;
end;
figDirectory = strcat(figDirectory, '\');



% Gets the direction of the leader's DR errors.
% Returns a random value if the user's input is invalid
function heading = getLVDRErrorDir
global GUIHandles;
heading = str2double(get(GUIHandles.leaderDirection, 'String'));
if(isnan(heading))
    heading = 0;
    set(GUIHandles.leaderDirection, 'String', num2str(heading));
end

% Gets the direction of the follower's DR errors.
% Returns a random value if the user's input is invalid
```

```
function heading = getFVDRErrorDir
global GUIHandles;
heading = str2double(get(GUIHandles.followerDirection, 'String'));
if(isnan(heading))
    heading = 0;
    set(GUIHandles.leaderDirection, 'String', num2str(heading));
end


% ********************************************************************
%           END OF GUI INTERFACE FUNCTIONS
% ********************************************************************
```

D. PRIMARY PROCESSING

1. Overview

The code provided in this section is the primary processing function, similar to the main function in other popular programming languages.

2. CorrectNavErrors.m

```
function CorrectNavErrors
% Author: Dan Kucik
%        Naval Postgraduate School
% CorrectNavErrors - Implements the simple relative navigation
% error correction algorithm.  This function is designed to operate
% with dataEntryGUI.m/.fig.
%
% General Notes:
% 1) The variables with names containing the word "true" refer to the
%    true position of the vehicle and therefore these values
%    include the accumulated dead reckoning errors.
% 2) The variables with names containing the word "intended" refer
%    the intended position of the vehicle, i.e. the estimated
%    position of the LV.  These values do not include the DR
%    errors because the vehicles are not aware of their own
%    accumulated errors while using dead reckoning.  These
%    are the values that would be reported during vehicle-
%    to-vehicle communications/position updates.


global generateSummaryPlots;
generateSummaryPlots = 1;
    %flag indicating if the comprehensive or summary plots should be
    %displayed/stored
    %=0 for complete / comprehensive plots
    %=1 for summary plots
    %=2 for all plots

global PCRelativeError;
    %the relative errors between the LV and FV before correction

global calculationMethod;
    %the calculation method to be used to determine the correction
    %factor
    % =1 for simple trig method
    % =2 for the processor intensive over-determined system of
    % non-linear distance equations
```

```
global speed;
    %speed of the LV vehicles before and during calibration (m/s)

global outputFile;
    %the name of the file to store the simulation data

global leaderDRErrorDir;
    %the direction of the lead vehicle's dead reckoning errors (degrees)
    %This is an absolute direction that is linearly applied to the vehicle's
    %position

global followerDRErrorDir;
    %the direction of the follower vehicle's dead reckoning errors (degrees)
    %This is an absolute direction that is linearly applied to the vehicle's
    %position

global leaderHeading;
    %the heading of the leader during the calibration (degrees)

global commsRangeDelay;
    %the delay between when the vehicle-to-vehicle comms is received and
    %when the ranging information is received (sec)

global preCorrectionDRTime;
    %the length of time the vehicles are dead reckoning before the start
    %of the calibration (sec).  This is used to determine the relative
    %navigation errors between the LV and the FV

global TPTransitTime;
    %the time required for the FV to transit between the each test point (sec)

global NumOfTPs;
    %the number of test points to use for the calibration

global PRangingError;
    %probability of a spurious ranging error (other than white noise)

global maxRangingError;
    %the maximum error associated with spurious ranging values

global testPoints;
    %the test points to be used for the calibration.  These are the positions
    %of the FV relative to the LV's position.  The format is (north,east,down)
    %so if these points are to be plotted where y/up is north remember to
    %adjust the ordering.

global LVTrueEndTPs;
    %LV's true position after each TP is completed (includes DR errors)

global FVTrueEndTPs;
    %FV's true position after each TP is completed (includes DR errors)

global figureNumber;
    %monitors the file names for the saved figures (increments starting at 1)

global firstFigure;
    %the first figure number (file name) for this test

global lastFigure;
    %the last figure number (file name) for this test

global figDirectory;
    %the directory that will hold the saved figures
```

```
global correctionFactor;
    %the calculated correction factor -- used to correct
    %the relative navigation errors between the LV and the FV

global totalErrorResidual;
    %The residual error after applying the correction factor

global errorResidual;
    %The components of the relative nav residual error

global totalImprovement;
    %The amount the error was reduced after the correction
    %factor was applied

global improvement;
    %the components of the error reduction resulting from
    %the use of the correction factor

global TPTransitTimes;
    %The time it takes to transit from one test point to
    %the next.

global percentReduction;
    %The percent improvement provided by the correction
    %factor

global displayPlots;
    %flag indicating if the plots should be displayed
    %=1 then plots are displayed

global savePlots;
    %flag indicating if the plots should be saved
    %=1 then plots are saved

global forcedError;
    %the forced baseline error used for a system of equations
    %approach

global trueRangesErrors;
    %the range errors

global autoErrorDir;
    %flag indicating if the user elected to cycle through
    %a series of 64 automatic tests where the LV and FV's
    %error directions are varied by increments of 45-deg

global numberOfCorrections;
    %the number of recurssive corrections to perform

global correctionCount;
correctionCount = 1;

if(calculationMethod == 1 | calculationMethod == 3)
    if(leaderDRErrorDir == -1 & followerDRErrorDir == -1)
        %if the user elected to cycle through
        %a series of 64 automatic tests where the LV and FV's
        %error directions are varied by increments of 45-deg

        autoErrorDir = 1;   %update flag to indicate running in auto mode

       for(leaderDRErrorDir = 0: -0.5: -3)
            for(followerDRErrorDir = 0: 0.5: 3)
```

```
                %run combinations of various error directions
                %with increments of 45-degrees

                 correctionCount = 1;    %current correction count
                 executeTwoPointCorrection
            end
        end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% start of code for user defined error directions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    else

        disp('running fixed bias mode');
        autoErrorDir = 0;   %update flag to indicate running not in auto mode

        correctionCount = 1;    %current correction count
        executeTwoPointCorrection

    end
end

% xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
% Start of processing system of equations method
% xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

if(calculationMethod==2)
    [LVIntendedTrack, LVIntendedStartTPs, LVIntendedEndTPs, LVTrueTrack,
LVTrueStartTPs, LVTrueEndTPs] = calculateLVTrack(testPoints, NumOfTPs,
leaderHeading, speed, leaderDRErrorDir, preCorrectionDRTime, TPTransitTime,
commsRangeDelay, forcedError);
            %calculate the track and position of the testpoints for the leader

    [FVIntendedTrack, FVIntendedStartTPs, FVIntendedEndTPs, FVTrueTrack,
FVTrueStartTPs, FVTrueEndTPs] = calculateFVTrack(testPoints, NumOfTPs,
leaderHeading, followerDRErrorDir, LVIntendedStartTPs, LVIntendedEndTPs,
commsRangeDelay);
            %calculate the track and position of the testpoints for the
follower


    [trueRanges, trueRangesErrors] = calculateRanges(LVTrueEndTPs,
FVTrueEndTPs, NumOfTPs, PRangingError, maxRangingError);
            %Calculate the range between the LV and FV for each testpoint

    [expectedRanges, trash] = calculateRanges(LVIntendedEndTPs,
FVIntendedEndTPs, NumOfTPs, 0, 0);
            %calculate the range between the LV and FV for each TP (incl DR
errors)


    correctionFactor = SOECorrectionFactorCalc(LVIntendedEndTPs,
FVIntendedEndTPs, trueRanges, leaderHeading, testPoints, NumOfTPs);

    [totalErrorResidual, errorResidual, totalImprovement, improvement,
percentReduction, PCRelativeError] = analyzeCorrectionFactor(correctionFactor,
LVTrueEndTPs(NumOfTPs,:),FVTrueEndTPs(NumOfTPs,:),
LVIntendedEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:))
            %generate some statistics for the correction factor
    firstFigure = figureNumber;
            %store the figure file name/number for later use in output file

    if(displayPlots == 1 | savePlots ==1)
                %if the user wishes to plot the data

        [LVTrue, LVIntended, FVTrueUncorrected, FVTrueCorrected,
FVIntendedUncorrected, FVIntendedAfterRelCor] =
generateComparisionTracks(leaderDRErrorDir, leaderHeading, followerDRErrorDir,
LVTrueEndTPs(NumOfTPs,:), LVIntendedEndTPs(NumOfTPs,:),
FVTrueEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:), TPTransitTime, 100,
500, correctionFactor);
            %generate a set of tracks that incorporates the correction factor

        generateSummaryTrackPlot(LVIntendedTrack, LVTrueTrack, FVIntendedTrack,
FVTrueTrack, figDirectory);

        generateSummaryTPPlot(LVIntendedStartTPs, LVIntendedEndTPs,
LVIntendedTrack, LVTrueStartTPs, LVTrueEndTPs, LVTrueTrack, FVIntendedStartTPs,
FVIntendedEndTPs, FVIntendedTrack, FVTrueStartTPs, FVTrueEndTPs, FVTrueTrack,
NumOfTPs, figDirectory);

        generateSummaryResultsPlot(LVTrue, LVIntended, FVTrueUncorrected,
FVTrueCorrected, FVIntendedUncorrected, FVIntendedAfterRelCor, figDirectory);

    end

    lastFigure = figureNumber-1;
            %save the file name for the last plot created for this run
            %this info will be stored in the output file

    outputDataToFile;
        %store the data to the output file
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% executeTwoPointCorrection
% Performs the high level functions of the calibration
% procedure that uses two points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function executeTwoPointCorrection()

global generateSummaryPlots;

global PCRelativeError;
    %the relative errors between the LV and FV before correction

global calculationMethod;
    %the calculation method to be used to determine the correction
    %factor
    % =1 for simple trig method
    % =2 for the processor intensive over-determined system of
    % non-linear distance equations

global speed;
    %speed of the LV vehicles before and during calibration (m/s)

global outputFile;
    %the name of the file to store the simulation data

global leaderDRErrorDir;
    %the direction of the lead vehicle's dead reckoning errors (degrees)
    %This is an absolute direction that is linearly applied to the vehicle's
    %position

global followerDRErrorDir;
    %the direction of the follower vehicle's dead reckoning errors (degrees)
    %This is an absolute direction that is linearly applied to the vehicle's
    %position

global leaderHeading;
    %the heading of the leader during the calibration (degrees)

global commsRangeDelay;
    %the delay between when the vehicle-to-vehicle comms is received and
    %when the ranging information is received (sec)

global preCorrectionDRTime;
    %the length of time the vehicles are dead reckoning before the start
    %of the calibration (sec).  This is used to determine the relative
    %navigation errors between the LV and the FV

global TPTransitTime;
    %the time required for the FV to transit between the each test point (sec)

global NumOfTPs;
    %the number of test points to use for the calibration

global PRangingError;
    %probability of a spurious ranging error (other than white noise)

global maxRangingError;
    %the maximum error associated with spurious ranging values

global testPoints;
    %the test points to be used for the calibration.  These are the positions
    %of the FV relative to the LV's position.  The format is (north,east,down)
    %so if these points are to be plotted where y/up is north remember to
    %adjust the ordering.

global LVTrueEndTPs;
    %LV's true position after each TP is completed (includes DR errors)

global FVTrueEndTPs;
    %FV's true position after each TP is completed (includes DR errors)

global figureNumber;
    %monitors the file names for the saved figures (increments starting at 1)

global firstFigure;
    %the first figure number (file name) for this test

global lastFigure;
    %the last figure number (file name) for this test

global figDirectory;
    %the directory that will hold the saved figures

global correctionFactor;
    %the calculated correction factor -- used to correct
    %the relative navigation errors between the LV and the FV

global totalErrorResidual;
    %The residual error after applying the correction factor
```

```
global errorResidual;
    %The components of the relative nav residual error

global totalImprovement;
    %The amount the error was reduced after the correction
    %factor was applied

global improvement;
    %the components of the error reduction resulting from
    %the use of the correction factor

global TPTransitTimes;
    %The time it takes to transit from one test point to
    %the next.

global percentReduction;
    %The percent improvement provided by the correction
    %factor

global displayPlots;
    %flag indicating if the plots should be displayed
    %=1 then plots are displayed

global savePlots;
    %flag indicating if the plots should be saved
    %=1 then plots are saved

global forcedError;
    %the forced baseline error used for a system of equations
    %approach

global trueRangesErrors;
    %the range errors

global autoErrorDir;
    %flag indicating if the user elected to cycle through
    %a series of 64 automatic tests where the LV and FV's
    %error directions are varied by increments of 45-deg

global numberOfCorrections;
    %the number of recurssive corrections to perform

global correctionCount;

correctionCount = 1;    %current correction count

[LVIntendedTrack, LVIntendedStartTPs, LVIntendedEndTPs, LVTrueTrack,
LVTrueStartTPs, LVTrueEndTPs] = calculateLVTrack(testPoints, NumOfTPs,
leaderHeading, speed, leaderDRErrorDir, preCorrectionDRTime, TPTransitTime,
commsRangeDelay, forcedError);
            %calculate the track and position of the testpoints for the leader

[FVIntendedTrack, FVIntendedStartTPs, FVIntendedEndTPs, FVTrueTrack,
FVTrueStartTPs, FVTrueEndTPs] = calculateFVTrack(testPoints, NumOfTPs,
leaderHeading, followerDRErrorDir, LVIntendedStartTPs, LVIntendedEndTPs,
commsRangeDelay);
            %calculate the track and position of the testpoints for the
follower


[trueRanges, trueRangesErrors] = calculateRanges(LVTrueEndTPs, FVTrueEndTPs,
NumOfTPs, PRangingError, maxRangingError);
            %Calculate the range between the LV and FV for each testpoint
```

```
[expectedRanges, trash] = calculateRanges(LVIntendedEndTPs, FVIntendedEndTPs,
NumOfTPs, 0, 0);
            %calculate the range between the LV and FV for each TP (incl DR
errors)



correctionFactor = simpleCorrectionFactorCalc(LVIntendedEndTPs,
FVIntendedEndTPs, trueRanges, leaderHeading, testPoints, NumOfTPs);
            %calculate the correction factor for reducing the relative
navigation errors
correctionCount
[totalErrorResidual, errorResidual, totalImprovement, improvement,
percentReduction, PCRelativeError] = analyzeCorrectionFactor(correctionFactor,
LVTrueEndTPs(NumOfTPs,:),FVTrueEndTPs(NumOfTPs,:),
LVIntendedEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:))
            %generate some statistics for the correction factor

firstFigure = figureNumber;
            %store the figure file name/number for later use in output file
if(displayPlots == 1 | savePlots ==1)
                %if the user wishes to plot the data

    [LVTrue, LVIntended, FVTrueUncorrected, FVTrueCorrected,
FVIntendedUncorrected, FVIntendedAfterRelCor] =
generateComparisionTracks(leaderDRErrorDir, leaderHeading, followerDRErrorDir,
LVTrueEndTPs(NumOfTPs,:), LVIntendedEndTPs(NumOfTPs,:),
FVTrueEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:), TPTransitTime, 100,
500, correctionFactor);
                %generate a set of tracks that incorporates the correction
factor

        generateSummaryTrackPlot(LVIntendedTrack, LVTrueTrack, FVIntendedTrack,
FVTrueTrack, figDirectory);

        generateSummaryTPPlot(LVIntendedStartTPs, LVIntendedEndTPs,
LVIntendedTrack, LVTrueStartTPs, LVTrueEndTPs, LVTrueTrack, FVIntendedStartTPs,
FVIntendedEndTPs, FVIntendedTrack, FVTrueStartTPs, FVTrueEndTPs, FVTrueTrack,
NumOfTPs, figDirectory);

        generateSummaryResultsPlot(LVTrue, LVIntended, FVTrueUncorrected,
FVTrueCorrected, FVIntendedUncorrected, FVIntendedAfterRelCor, figDirectory);

end

lastFigure = figureNumber-1;
    %save the file name for the last plot created for this run
    %this info will be stored in the output file

outputDataToFile;
    %store the data to the output file


for(correctionCount = 2: numberOfCorrections)

    correctionCount

    [LVIntendedTrack, LVIntendedStartTPs, LVIntendedEndTPs, LVTrueTrack,
LVTrueStartTPs, LVTrueEndTPs] = calculateLVTrack(testPoints, NumOfTPs,
leaderHeading, speed, leaderDRErrorDir, TPTransitTime, TPTransitTime,
commsRangeDelay, -errorResidual);
        %calculate the track and position of the testpoints for the leader
    correctionMatrix = [correctionFactor;correctionFactor];
        %correction factor in matrix form to correct the values calculated by
the LV

    [FVIntendedTrack, FVIntendedStartTPs, FVIntendedEndTPs, FVTrueTrack,
FVTrueStartTPs, FVTrueEndTPs] = calculateFVTrack(testPoints, NumOfTPs,
leaderHeading, followerDRErrorDir, LVIntendedStartTPs+correctionMatrix,
LVIntendedEndTPs+correctionMatrix, commsRangeDelay);
        %calculate the track and position of the testpoints for the follower


    [trueRanges, trueRangesErrors] = calculateRanges(LVTrueEndTPs,
FVTrueEndTPs, NumOfTPs, PRangingError, maxRangingError);
        %Calculate the range between the LV and FV for each testpoint

    [expectedRanges, trash] = calculateRanges(LVIntendedEndTPs,
FVIntendedEndTPs, NumOfTPs, 0, 0);
        %calculate the range between the LV and FV for each TP (incl DR errors)

    correctionFactor = simpleCorrectionFactorCalc(LVIntendedEndTPs,
FVIntendedEndTPs, trueRanges, leaderHeading, testPoints, NumOfTPs);
        %calculate the correction factor for reducing the relative navigation
errors

    [totalErrorResidual, errorResidual, totalImprovement, improvement,
percentReduction, PCRelativeError] = analyzeCorrectionFactor(correctionFactor,
LVTrueEndTPs(NumOfTPs,:),FVTrueEndTPs(NumOfTPs,:),
LVIntendedEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:))
        %generate some statistics for the correction factor

    firstFigure = figureNumber;
         %store the figure file name/number for later use in output file
    if(displayPlots == 1 | savePlots ==1)
                 %if the user wishes to plot the data

          [LVTrue, LVIntended, FVTrueUncorrected, FVTrueCorrected,
FVIntendedUncorrected, FVIntendedAfterRelCor] =
generateComparisionTracks(leaderDRErrorDir, leaderHeading, followerDRErrorDir,
LVTrueEndTPs(NumOfTPs,:), LVIntendedEndTPs(NumOfTPs,:),
FVTrueEndTPs(NumOfTPs,:), FVIntendedEndTPs(NumOfTPs,:), TPTransitTime, 100,
500, correctionFactor);
                    %generate a set of tracks that incorporates the correction
factor

              generateSummaryTrackPlot(LVIntendedTrack, LVTrueTrack,
FVIntendedTrack, FVTrueTrack, figDirectory);

              generateSummaryTPPlot(LVIntendedStartTPs, LVIntendedEndTPs,
LVIntendedTrack, LVTrueStartTPs, LVTrueEndTPs, LVTrueTrack, FVIntendedStartTPs,
FVIntendedEndTPs, FVIntendedTrack, FVTrueStartTPs, FVTrueEndTPs, FVTrueTrack,
NumOfTPs, figDirectory);

              generateSummaryResultsPlot(LVTrue, LVIntended, FVTrueUncorrected,
FVTrueCorrected, FVIntendedUncorrected, FVIntendedAfterRelCor, figDirectory);
    end
    lastFigure = figureNumber-1;
          %save the file name for the last plot created for this run
          %this info will be stored in the output file
    outputDataToFile;
                %store the data to the output file
end
```

E. CORRECTION FACTOR CALCULATION USING TWO RANGE VALUES

1. Overview

The code in this section calculates the relative-navigation error-correction factor using the ranging data collected at two TPs.

2. SimpleCorrectionFactorCalc.m

```
function correctionFactor = simpleCorrectionFactorCalc(LVIntendedEndTPs,
FVIntendedEndTPs, trueRanges, LVHeading, testPoints, NumOfTPs)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% simpleCorrectionFactorCalc calculates the correction factor that the FV
% must add to the positions received from the LV to correct for the
% accumulated DR navigation errors.
% This is the simplified/least processor intensive version of the calculation
% because it does not utilize a system of equations.
%
% Special note about the calibration reference frame
% The origin is at TP2
% The x-axis is in the direction of the baseline passing through TP2
% The y-axis is perpendicular to the baseline and passes through TP2
%
%
%        TP2
%           \
%          B \
%           a \
%            s \
%             e \
%              l \
%               i \
%                n \
%                 e \
%                    TP1
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global calculationMethod;

rotatedTPs = rotateRefFrame(LVHeading, testPoints, NumOfTPs);
    %rotate the original testPoints to match the LV heading

if(calculationMethod ==3)   %if using the parallel method
    TPConfig = determineTPConfigForParallel(testPoints);
        %determine the configuration of the test points
        %so the rotation can be performed correctly
end


baseline = range(testPoints(1,:), testPoints(2,:));
    % calculate the baseline length
    % note this mimics the true case because the baseline would
    % be calculated using the true values because the FV is
    % not aware of its own errors.

tempX = (trueRanges(2)^2 + baseline^2 - trueRanges(1)^2)/(2*baseline);
    % calculate the X Position of the LV relative to TP2
    % note this is in the temp calibration reference frame

tempY = sqrt(abs(trueRanges(2)^2 - tempX^2));
    % calculate the Y Position of the LV relative to TP2
    % note this is in the temp calibration reference frame
    % note that the LV must be in front of the FV/baseline otherwise
    % the sign will be incorrect

LVPosInCorrectionFrame = [tempX, tempY, LVIntendedEndTPs(2,3)];
    %the position of the LV relative to TP2 in the correction
    %frame.

if((calculationMethod == 3 & (TPConfig == 1 | TPConfig == 4)) |
(calculationMethod ==1 & (testPoints(2,2) > testPoints(1,2))))
    %special case where TP2 is located NE or SW of LV,
    %which requires a different rotation method
    %                       ^ +N
    %                       |
    %                       |
    %                       |
    %                       |
    %                       |
    %                       O--------------------> +E
    %
    %
    %              TP1             TP2
    disp('special TP case');
    correctionFrameXAxisAng = (atan2((rotatedTPs(1,2) -
rotatedTPs(2,2)),(rotatedTPs(1,1) - rotatedTPs(2,1))));
        %Angle the correction frame's x-axis makes with the N north axis

    LVPosInGlobalFrameRelToTP2 =
rotateRefFrame((rad2deg(correctionFrameXAxisAng)), LVPosInCorrectionFrame, 1);
        %rotate the LV position from the correction frame back
        %to the N-E-D frame such that the correction frame's
        %x and y axes line up with the global frame's N and E
        %north axes respectively



else    %the user selected the follow-the-leader approach or
        %TP2 is located NW or SE of the LV
        %                       ^ +N
        %                       |
        %                       |
        %                       |
        %                       |
        %                       |
        %                       O--------------------> +E
        %
        %
        %              TP2             TP1


    correctionFrameXAxisAng = (atan2((rotatedTPs(1,1) - rotatedTPs(2,1)),
(rotatedTPs(1,2) - rotatedTPs(2,2))));
    %Angle the correction frame's x-axis makes with the N-E-D frame

    LVPosInGlobalFrameRelToTP2 =
rotateRefFrame((rad2deg(correctionFrameXAxisAng)), LVPosInCorrectionFrame, 1);
        %rotate the LV position from the correction frame back
        %to the N-E-D frame such that the correction frame's
        %x and y axes line up with the global frame's east and
        %north axes respectively

    LVPosInGlobalFrameRelToTP2 = [LVPosInGlobalFrameRelToTP2(2),
LVPosInGlobalFrameRelToTP2(1),LVPosInGlobalFrameRelToTP2(3)];
        %it is necessary to flip the x and y to account for
        %the correction frame's x and y axes being rotated to
        % line up with the global frame's east and
        %north axes respectively
end


LVIntendedPosRelToTP2 = LVIntendedEndTPs(2,:) - FVIntendedEndTPs(2,:);
    %The intended position of the LV relative to the FV based on
    %vehicle-to-vehicle communications (excludes DR errors)

correctionFactor = LVPosInGlobalFrameRelToTP2 - LVIntendedPosRelToTP2;
    %calculate the correction factor, which will later be added to
    %the leader position updated to minimize the relative error

correctionFactor(3) = 0;
    %This has been simplified to a 2-D problem because
    %pressure/depth sensors are accurate since the
    %vehicles are operating in the same environment
```

F. CORRECTION FACTOR CALCULATION USING THREE OR MORE RANGE VALUES

1. Overview

The code in this section calculates the relative-navigation error-correction factor for the case where three or more TPs are used.

2. SOECorrectionFactorCalc.m

```
function correctionFactor = SOECorrectionFactorCalc(LVIntendedEndTPs,
FVIntendedEndTPs, trueRanges, LVHeading, testPoints, NumOfTPs)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOECorrectionFactorCalc - uses a system of distance equations
% to calculate the relative navigation error correction factor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rotatedTPs = rotateRefFrame(LVHeading, testPoints, NumOfTPs);
    %rotate the original testPoints to match the LV heading

pos0 = [10; 10];    %initial guess
options = optimset('MaxIter', 10000000);
[LVPositionRelToArtOrigin,fval] = fsolve(@positionTriangulationEqn, pos0,
options, NumOfTPs, rotatedTPs, trueRanges, LVIntendedEndTPs);
    %LVPositionRelToArtOrgin now contains the position of the LV relative to
the
    %artifical origin used to enter the test points but after the rotation to
    %match the LV's heading.

correctionFactor=[LVPositionRelToArtOrigin(1), LVPositionRelToArtOrigin(2),0];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% positionTriangulationEqn - defines the equations for the
% triangulation of a vehicle.
```

```
%
% Param Descriptions
% pos = the iterative positions tested in the equation (generated by
%       fsolve
% NumOfTPs = The number of test points used for the calibration (3-10)
% testPoints = a NumOfTPs x 3 array containing the testpoints for the
%              the calibration
%              NOTE these positions should be the north east down
%              in global coordinates of the FV relative to the LV.
%              column 1 = north position
%              column 2 = east position
%              column 3 = down position
% ranges = the range values obtained from acoustic ranging
%          this should be a NumOfTPs x 1 array where each row
%          is the range obtained for each test point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function eqn = positionTriangulationEqn(pos, NumOfTPs, testPoints, ranges,
LVIntendedPositions)
eqn = [];
for(k=1:NumOfTPs)
    eqn = [eqn; (testPoints(k,1) - pos(1))^2 + (testPoints(k,2) - pos(2))^2 +
(testPoints(k,3))^2  - ranges(k)^2];
end
```

G. LV TRACK CALCULATION

1. Overview

The code in this section calculates the LV’s track based on the parameters entered into the GUI.

2. CalculateLVTrack.m

```
function [intendedTrack, intendedStartTPs, intendedEndTPs, trueTrack,
trueStartTPs, trueEndTPs] = calculateLVTrack(testPoints, NumOfTPs,
leaderHeading, speed, leaderDRErrorDir, preCorrectionDRTime, TPTransitTime,
commsRangeDelay, forcedError)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculates the position of the LV for each test point.
% It is assumed that the vehicle started at 0,0
%
% intendedTrack - the planned track for the LV (w/o DR errors)
% intendedStartTPs - the intended position of the LV at the
%                    beginning of the test points (w/o DR errors)
% intendedEndTPs - the intended position of the LV at the
%                  end of the test points (w/o DR errors)
% trueTrack - The track/path taken by the LV (w/ DR errors)
% trueStartTPs - the true position of the LV at the
%                beginning of the test points (w/ DR errors)
% trueEndTPs - the true position of the LV at the
%              end of the test points (w/ DR errors)
%
% testPoints = matrix containing the test points for the cal.
%  These points are in the N-E-D frame and are relative to the
%  LV as if it were at (0,0,0) and pointing north.
% NumOfTPs = the number of test points to be used for the cal.
% leaderHeading = the heading of the LV during the cal (in degs)
% speed = the speed of the LV in m/s
% leaderDRErrorDir = the direction of the DR error relative to
% to the NED coordinate system (0 deg = north, 90 deg = east, ..
% preCorrectionDRTime = the length of time the LV was using DR
%  before starting the calibration
% TPTransitTime - the amount of time it takes to reach the
%  next test point.
% commsRangeDelay - The delay between when the FV receives
%  position update from the LV and when the ranging values
%  are received.  In seconds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
heading = deg2rad(leaderHeading);   %LV heading in rads

errorHeading = deg2rad(leaderHeading + leaderDRErrorDir);
    %LV DR error direction in rads


%both the intended and the true start at 0,0 before the LV starts DRing
intendedTrack = [0,0,0];
trueTrack = forcedError;

%calculate the start position (excluding DR errors) of the first TP
tempX = ((speed * preCorrectionDRTime) * cos(heading));
tempY = ((speed * preCorrectionDRTime) * sin(heading));
intendedTrack = [intendedTrack; tempX, tempY, testPoints(1,3)];
intendedStartTPs = [tempX, tempY, testPoints(1,3)];

%calculate the end of the first intended TP
temp2X = tempX + (commsRangeDelay * speed * cos(heading));
temp2Y = tempY + (commsRangeDelay * speed * sin(heading));
intendedTrack = [intendedTrack; temp2X, temp2Y, testPoints(1,3)];
intendedEndTPs = [temp2X, temp2Y, testPoints(1,3)];

%calculate the start position of the first test point (including DR errors)
tempX = ((speed * preCorrectionDRTime) * cos(errorHeading)) + forcedError(1);
tempY = ((speed * preCorrectionDRTime) * sin(errorHeading)) + forcedError(2);
trueTrack = [trueTrack; tempX, tempY, testPoints(1,3)];
trueStartTPs = [tempX, tempY, testPoints(1,3)];

%calculate the end position of the first test point (including DR errors)
temp2X = tempX + (commsRangeDelay * speed * cos(errorHeading));
temp2Y = tempY + (commsRangeDelay * speed * sin(errorHeading));
trueTrack = [trueTrack; temp2X, temp2Y, testPoints(1,3)];
trueEndTPs = [temp2X, temp2Y, testPoints(1,3)];

for(k=2:NumOfTPs) %starting at k=2 since first TP location already calculated
    %calculate the intended start point for the next TP

    tempX = intendedEndTPs(k-1,1) + ((speed * TPTransitTime) * cos(heading));
    tempY = intendedEndTPs(k-1,2) + ((speed * TPTransitTime) * sin(heading));
    intendedTrack = [intendedTrack; tempX, tempY, testPoints(k,3)];
    intendedStartTPs = [intendedStartTPs; tempX, tempY, testPoints(k,3)];

    %calculate the intended end point for the TP
    temp2X = tempX + (commsRangeDelay * speed * cos(heading));
    temp2Y = tempY + (commsRangeDelay * speed * sin(heading));
    intendedTrack = [intendedTrack; temp2X, temp2Y, testPoints(k,3)];
    intendedEndTPs = [intendedEndTPs; temp2X, temp2Y, testPoints(k,3)];

    %calculate the true vehicle position for each TP (including DR errors)
    tempX = trueEndTPs(k-1,1) + ((speed * TPTransitTime) * cos(errorHeading));
    tempY = trueEndTPs(k-1,2) + ((speed * TPTransitTime) * sin(errorHeading));
    trueTrack = [trueTrack; tempX, tempY, testPoints(k,3)];
    trueStartTPs = [trueStartTPs; tempX, tempY, testPoints(k,3)];

    %end of TP (w/ dr errors)
    temp2X = tempX + (commsRangeDelay * speed * cos(errorHeading));
    temp2Y = tempY + (commsRangeDelay * speed * sin(errorHeading));
    trueTrack = [trueTrack; temp2X, temp2Y, testPoints(k,3)];
    trueEndTPs = [trueEndTPs; temp2X, temp2Y, testPoints(k,3)];
end
```

H. FV TRACK CALCULATION

1. Overview

The code in this section calculates the FV's track based on the parameters entered into the GUI and the data generated by calculateLVTrack.m.

2. CalculateFVTrack.m

```
function [intendedTrack, intendedStartTPs, intendedEndTPs, trueTrack,
trueStartTPs, trueEndTPs] = calculateFVTrack(testPoints, NumOfTPs,
leaderHeading, followerDRErrorDir, LVIntendedStartTPs, LVIntendedEndTPs,
commsRangeDelay)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculateFVTrack - Calculates the position of the FV for each test point.
% It is assumed that the vehicle started at 0,0.
%
% intendedTrack - the planned track for the FV (w/o DR errors)
% intendedStartTPs - the intended position of the FV at the
%                    beginning of the test points (w/o DR errors)
% intendedEndTPs - the intended position of the FV at the
%                  end of the test points (w/o DR errors)
% trueTrack - The track/path taken by the FV (w/ DR errors)
% trueStartTPs - the true position of the FV at the
%                beginning of the test points (w/ DR errors)
% trueEndTPs - the true position of the FV at the
%              end of the test points (w/ DR errors)
%
% testPoints = matrix containing the test points for the cal.
%  These points are in the N-E-D frame and are relative to the
%  LV as if it were at (0,0,0) and pointing north.
% NumOfTPs = the number of test points to be used for the cal.
% leaderHeading = the heading of the LV during the cal (in degs)
% followerDRErrorDir = the direction of the DR error relative to
% to the NED coordinate system (0 deg = north, 90 deg = east, ..
% LVIntendedStartTPs - The LV intended position for the start
%  of each test point.  This would be received from the LV
%  position update sent over the acoustic modem.  Note that
%  the DR errors still accumulate between the time the FV
%  receives the LV position update and the time the ranging
%  information is received.
% LVIntendedEndTPs - The LV position at the end of each
%  test point.  This information would really be calculated
%  using the LV's position at the start of the test point,
%  its heading, and speed.
% commsRangeDelay - The delay between when the FV receives
%  position update from the LV and when the ranging values
%  are received.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
errorHeading = deg2rad(followerDRErrorDir);
    %the LV error heading in rads

rotatedTPs = rotateRefFrame(leaderHeading, testPoints, NumOfTPs);
    %the test points rotated to match the LV's heading
```

```
%assume the FV starts at 0,0,0 when it begins DRing
intendedTrack = [0,0,0];
trueTrack = [0,0,0];
intendedStartTPs = [];
intendedEndTPs = [];
trueStartTPs = [];
trueEndTPs = [];

targetPointX2 = 0;
targetPointY2 = 0;
targetPointX4 = 0;
targetPointY4 = 0;

temp2X = 0;     % the current x position of the FV
temp2Y = 0;     % the current y position of the FV
dist2 = 0;      % temp var to calculate distance traveled and
                %holds value to carry over to later iterations.

for(k=1:NumOfTPs)
    %calculated the intended start position for the first TP
    targetPointX1 = LVIntendedStartTPs(k,1) + rotatedTPs(k,1);
    targetPointY1 = LVIntendedStartTPs(k,2) + rotatedTPs(k,2);
    [m,n] = size(intendedTrack);
    angleToTarget1 = atan2((targetPointY1 - intendedTrack(m,2)), (targetPointX1
- intendedTrack(m,1)));
    distanceToTarget1 = sqrt((targetPointX1 - intendedTrack(m,1))^2 +
(targetPointY1 - intendedTrack(m,2))^2);
    tempX1 = intendedTrack(m,1) + (distanceToTarget1 * cos(angleToTarget1));
    tempY1 = intendedTrack(m,2) + (distanceToTarget1 * sin(angleToTarget1));
    intendedTrack = [intendedTrack; tempX1, tempY1, testPoints(k,3)];
    intendedStartTPs = [intendedStartTPs; tempX1, tempY1, testPoints(k,3)];

    %calculate the end of the first intended TP
    targetPointX2 = LVIntendedEndTPs(k,1) + rotatedTPs(k,1);
    targetPointY2 = LVIntendedEndTPs(k,2) + rotatedTPs(k,2);
    [m,n] = size(intendedTrack);
    angleToTarget2 = atan2((targetPointY2 - intendedTrack(m,2)), (targetPointX2
- intendedTrack(m,1)));
    distanceToTarget2 = sqrt((targetPointX2 - intendedTrack(m,1))^2 +
(targetPointY2 - intendedTrack(m,2))^2);
    tempX2 = intendedTrack(m,1) + (distanceToTarget2 * cos(angleToTarget2));
    tempY2 = intendedTrack(m,2) + (distanceToTarget2 * sin(angleToTarget2));
    intendedTrack = [intendedTrack; tempX2, tempY2, testPoints(k,3)];
    intendedEndTPs = [intendedEndTPs; tempX2, tempY2, testPoints(k,3)];

    %calculate the start position of the first test point (including DR errors)
    angleToTarget3 = angleToTarget1 + errorHeading;
    [m,n] = size(trueTrack);
    tempX3 = trueTrack(m,1) + (distanceToTarget1 * cos(angleToTarget3));
    tempY3 = trueTrack(m,2) + (distanceToTarget1 * sin(angleToTarget3));
    trueTrack = [trueTrack; tempX3, tempY3, testPoints(1,3)];
    trueStartTPs = [trueStartTPs; tempX3, tempY3, testPoints(k,3)];

    %calculate the end position of the first test point (including DR errors)
    angleToTarget4 =  angleToTarget2 + errorHeading;
    [m,n] = size(trueTrack);
    tempX4 = trueTrack(m,1) + (distanceToTarget2 * cos(angleToTarget4));
    tempY4 = trueTrack(m,2) + (distanceToTarget2 * sin(angleToTarget4));
    trueTrack = [trueTrack; tempX4, tempY4, testPoints(k,3)];
    trueEndTPs = [trueEndTPs; tempX4, tempY4, testPoints(k,3)];
end
```

I. CORRECTION FACTOR ANALYSIS

1. Overview

The code in this section performs an analysis of the calculated relative-navigation error-correction factor to determine its performance

2. AnalyzeCorrectionFactor.m

```
function [totalErrorResidual, errorResidual, totalImprovement, improvement,
percentReduction, PCRelativeError] =
analyzeCorrectionFactor(correctionFactor,LVFinalTruePoint,FVFinalTruePoint,
LVFinalIntendedPoint, FVFinalIntendedPoint)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% analyzeCorrectionFactor - analyzes the correction factor to determine the
% residual errors, adds to the vehicle tracks to included the correction
factor,
% and calculates the net improvement provided by the correction factor.
%
% totalErrorResidual - Magnitude of the relative errors after completing the
correction
% errorResidual - Components of the relative errors after completing the
correction
% totalImprovement - The reduction of relative nav errors resulting from the
correction
% improvement - components of the reduction of relative nav errors resulting
from the correction
% percentReduction - the percentage of relative errors removed by the
calibration
% PCRelativeError - the relative errors before the correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LVTruePosRelToFV = LVFinalTruePoint-FVFinalTruePoint;
    %The position of the LV relative to the FV including accumulated DR errors

LVIntendedPosRelToFV = LVFinalIntendedPoint - FVFinalIntendedPoint;
    %The intended position of the LV relative to the FV (excludes DR errors)

PCRelativeError = sqrt((LVTruePosRelToFV(1) - LVIntendedPosRelToFV(1))^2 +
(LVTruePosRelToFV(2) - LVIntendedPosRelToFV(2))^2);
    %Cacluation of the relative errors before the correction

LVPosRelToLVAfterCorrection = LVIntendedPosRelToFV + correctionFactor;
    %The relative position of the LV to the FV after the correction

errorResidual = LVTruePosRelToFV - LVPosRelToLVAfterCorrection;
    %the remaining relative errors after the correction

totalErrorResidual = sqrt(errorResidual(1)^2 + errorResidual(2)^2);
    %the total remaining relative error after completing the correction

relativeErrorPreCorrection = LVTruePosRelToFV - LVIntendedPosRelToFV;
    %The relative error between the LV and FV before the correction

relativeErrorPreCorrectionMag = sqrt(relativeErrorPreCorrection(1)^2 +
relativeErrorPreCorrection(2)^2);
    %The magitude of the relative error before the correction

totalImprovement = relativeErrorPreCorrectionMag - totalErrorResidual;
    %The net improvement provided by the correction
```

```
improvement = relativeErrorPreCorrection - errorResidual;
    %the components of the improvement provided by the correction

if(relativeErrorPreCorrectionMag == 0)
    percentReduction = 0;
else
    percentReduction = (totalImprovement/relativeErrorPreCorrectionMag) * 100;
        %the percentage improvement provided by the correction
end
```

J. SIMULATION RESULTS OUTPUT FILE

1. Overview

The code in this section outputs the simulation data to a comma-delimited text file for later analysis.

2. OutputDataToFile.m

```
function outputDataToFile
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%outputDataToFile - outputs the configuration and analysis information
%to the data file (outputFile) in a comma delimited format.
% Note that this routine uses global params because of the large number
% of values to be stored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global outputFile;
global figDirectory;
global calculationMethod;
global speed;
global leaderDRErrorDir;
global followerDRErrorDir;
global LVDRErrorRate;
global FVDRErrorRate;
global leaderHeading;
global commsRangeDelay;
global preCorrectionDRTime;
global TPTransitTime;
global NumOfTPs;
global PRangingError;
global maxRangingError;
global testPoints;
global LVTrueEndTPs
global FVTrueEndTPs
global figureNumber;
global firstFigure;
global lastFigure;
global figDirectory;
global autoErrorDir;
global totalErrorResidual;
global errorResidual;
global totalImprovement;
global improvement;
global percentReduction;
global PCRelativeError;
global forcedError;
global trueRangesErrors;
global correctionCount;
    timeToCompleteCorrection = (NumOfTPs * commsRangeDelay) + ((NumOfTPs -
1)*TPTransitTime);
```

```
        %rough calculate of the time required to complete the calibration

if(exist(outputFile) == 0) %if file is new
    %if the file doesn't already exit

    fid = fopen(outputFile, 'w');
        %open the file as a new file in write mode

    if(fid == -1)   %output msg if error opening file
        disp(strcat('Error opening new output file: ', outputFile));
    end

    %output general information
    fprintf(fid, 'Data File Name = %s\n', outputFile);
    fprintf(fid, 'Figure Directory: %s\n', figDirectory);
    fprintf(fid, 'Date: %s\n\n', date);

    %output the data format
    fprintf(fid, 'Data Format\n');
    fprintf(fid, 'Pre-Correction Error, Total Error Residual (m), North
Residual Error (m), East Residual Error (m), Total Improvment (m), North
Improvement (m), East Improvement (m), ');
    fprintf(fid, 'First Figure Number, Last Figure Number, Time to Complete
Correction, percent improvement, LV Speed (m/s), LV Compass Bias (Deg), FV
Compass Bias (Deg), ');
    fprintf(fid, 'Leader Heading (deg), Comms ranging delay(sec), DR Time
before correction (sec), Time between test points (sec), number of test points,
');
    fprintf(fid, 'Prob of spurious ranging, max ranging error (m), calculation
method, TP1N, TP1E, TP1D, TP2N, TP2E, TP2D, TP3N, TP3E, TP3D, TP4N, TP4E, TP4D,
');
    fprintf(fid, 'TP5N, TP5E, TP5D, TP6N, TP6E, TP6D, TP7N, TP7E, TP7D, TP8N,
TP8E, TP8D, TP9N, TP9E, TP9D, TP10N, TP10E, TP10D, True Range Error 1 (TRE1),
TRE2, TRE3, ');
    fprintf(fid, 'TRE4, TRE5, TRE6, TRE7, TRE8 , TRE9, TRE10, Forced Error N,
Forced Error E, Forced Error Down, correction count\n');

else    %if file already exists

    fid = fopen(outputFile, 'a');
        %open the file with append rights

    if(fid == -1)   %output an error msg if a file error occurred
        disp(strcat('Error opening output file: ', outputFile, ' for append'));
    end;
end;

%output the data in the above indicated format
fprintf(fid, '%f, %f, %f, %f, ', PCRelativeError, totalErrorResidual,
errorResidual(1), errorResidual(2));
fprintf(fid, '%f, %f, %f, ', totalImprovement, improvement(1), improvement(2));
fprintf(fid, '%i, %i, %f, %f, ', firstFigure, lastFigure,
timeToCompleteCorrection, percentReduction);
fprintf(fid, '%f, %f, %f, %f, %f, ', speed, leaderDRErrorDir,
followerDRErrorDir, LVDRErrorRate, FVDRErrorRate);
fprintf(fid, '%f, %f, %f, %f, %i, ', leaderHeading, commsRangeDelay,
preCorrectionDRTime, TPTransitTime, NumOfTPs);
fprintf(fid, '%f, %f, ', PRangingError, maxRangingError);
if(calculationMethod == 1)
    fprintf(fid, 'Simple Calc');
else
    fprintf(fid, 'Long SOE Calc');
end
```

```
for(k=1:10)
    if(k<=NumOfTPs)
        fprintf(fid, ', %f, %f, %f', testPoints(k,1), testPoints(k,2),
testPoints(k,3));
    else
        fprintf(fid, ', 0, 0, 0');
    end
end

for(k=1:10)
    if(k<=NumOfTPs)
        fprintf(fid, ', %f', trueRangesErrors(k));
    else
        fprintf(fid, ', 0');
    end
end


fprintf(fid,', %f, %f, %f, %i\n', forcedError(1), forcedError(2),
forcedError(3), correctionCount);

%if running automatic mode (where error directions automatically
%updated, then include separation between each series of runs
if((autoErrorDir == 1) & (leaderDRErrorDir == 315 & followerDRErrorDir == 315))
    fprintf(fid,'\n  \n  \n  \n  \n ');
end;

fclose(fid);    %close the data file
```

K. CORRECTION FACTOR PERFORMANCE TRACK GENERATION

1. Overview

The code in this section generates a set of tracks for the LV and FV to demonstrate the performance of the correction algorithm.

2. GenerateComparisionTracks.m

```
function [LVTrue, LVIntended, FVTrueUncorrected, FVTrueCorrected,
FVIntendedUncorrected, FVIntendedAfterRelCor] =
generateComparisionTracks(LVDRErrorDir, LVHeading, FVDRErrorDir,
LVLastTruePoint,LVLastIntendedPoint, FVLastTruePoint, FVLastIntendedPoint,
TPTransitTime, followingDistance, travelDistance, correctionFactor)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generateComparisonTracks - generates a set of tracks to show the
% impact/improvement provided by the correction factor.
% Note that it is assumed that the DR errors are linear.
% The resulting track has the following characteristics
% 1) The LV drives on the heading entered into the GUI for a distance of
travelDistance
% 2) The DR errors accumulated before, during, and after the correction
procedure
%    are included
% 3) The following vehicle attempts to follow the LV at a distance of
followingDistance
%
% Outputs
% LVTrue - The LV's true position in the global frame including accumulated DR
errors
% LVIntended - The path the LV intended to follow excluding DR errors
% FVTrueUncorrected - The true path (w/ DR errors) of the FV without
corrections
% FVTrueCorrected - The true path (w/DR errors) of the FV after the correction
%   factor was applied
% FVIntendedUncorrected - the intended path (w/o DR errors) of the FV without
%   the correction factor
% FVIntendedAfterRelCor - the intended path (w/o DR errors) of the FV with
%   the correction factor
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

heading = deg2rad(LVHeading);   %LV heading in rads

LVErrorHeading = deg2rad(LVHeading + LVDRErrorDir);
    %LV DR Error Direction

FVErrorHeading = deg2rad(LVHeading + FVDRErrorDir);
    %FV DR Error Direction

LVErrorsAfterTPs = LVLastTruePoint - LVLastIntendedPoint;
    %LV absolute error at the time the last test point
    %was completed



FVErrorsAfterTPs = FVLastTruePoint - FVLastIntendedPoint;
    %The FV's absolute error when the last test point
    %was completed

LVIntended = [LVLastIntendedPoint; (LVLastIntendedPoint(1) + (travelDistance *
cos(heading))), (LVLastIntendedPoint(2) + (travelDistance * sin(heading))),
LVLastIntendedPoint(3)];
    %start the track at the end of the last TP and add the next
    %point to the list

LVTrue = [LVLastTruePoint; (LVLastTruePoint(1) + (travelDistance *
cos(LVErrorHeading))), (LVLastTruePoint(2) + (travelDistance *
sin(LVErrorHeading))), LVLastTruePoint(3)];
    %the true track for the LV including DR errors.


FVIntendedUncorrected = [(LVIntended(1,1) - (followingDistance*cos(heading))),
(LVIntended(1,2) - (followingDistance*sin(heading))), FVLastIntendedPoint(3)];
FVIntendedUncorrected = [FVIntendedUncorrected; LVIntended(2,1) -
(followingDistance*cos(heading)), LVIntended(2,2) -
(followingDistance*sin(heading)), FVLastIntendedPoint(3)];
    %the FV intended (no DR errors) points based on the position of the LV


FVTrueUncorrected = [FVIntendedUncorrected(1,1) + FVErrorsAfterTPs(1),
FVIntendedUncorrected(1,2)+FVErrorsAfterTPs(2), FVIntendedUncorrected(1,3)];
FVTrueUncorrected = [FVTrueUncorrected; FVTrueUncorrected(1) + (travelDistance
* cos(FVErrorHeading)), FVTrueUncorrected(2) + (travelDistance *
sin(FVErrorHeading)), FVIntendedUncorrected(2,3)];
    %calculate the FV's true position by adding the accumulated
    %DR errors to the intended points

% The following lines are used to represent how the FV controller would use
% the correction factor.
% Basic idea is
%   1) Receive a position update from the LV
%   2) Add the correction factor to the position received from the LV
%   3) Calculate the desired position of the FV relative to the new corrected
LV position
```

```
correctionFactorMat = [correctionFactor;correctionFactor];  %setup correction
matrix
LVIntendedAfterRelCor = LVIntended + correctionFactorMat;   %correct the
position received from the LV



FVIntendedAfterRelCor = [(LVIntendedAfterRelCor(1,1) -
(followingDistance*cos(heading))), (LVIntendedAfterRelCor(1,2) -
(followingDistance*sin(heading))), FVLastIntendedPoint(3)];
FVIntendedAfterRelCor = [FVIntendedAfterRelCor; LVIntendedAfterRelCor(2,1) -
(followingDistance*cos(heading)), LVIntendedAfterRelCor(2,2) -
(followingDistance*sin(heading)), FVLastIntendedPoint(3)];
    %Note: the above calculation for FVIntendedAfterRelCor is identical
    %to the calculation of FVIntended except the corrected position of
    %the LV is used


FVTrueCorrected = [FVIntendedAfterRelCor(1,1) + FVErrorsAfterTPs(1),
FVIntendedAfterRelCor(1,2)+FVErrorsAfterTPs(2), FVLastIntendedPoint(3)];
FVTrueCorrected = [FVTrueCorrected; FVTrueCorrected(1) + (travelDistance *
cos(FVErrorHeading)), FVTrueCorrected(2) + (travelDistance *
sin(FVErrorHeading)), FVLastIntendedPoint(3)];
```

L. GRAPH GENERATION

1. Overview

The code in this section generates a series of plots showing the behavior of the vehicles before and during the relative-navigation correction procedure, and plots the data performance data calculated using generateComparisonTracks.m

2. GenerateSummaryTrackPlot.m

```
function generateSummaryTrackPlot(LVIntendedTrack, LVTrueTrack,
FVIntendedTrack, FVTrueTrack, figDirectory)
%generates two plots showing the LV and FV tracks
%before and during the correction.

global figureNumber;
global displayPlots;
global savePlots;


figure
hold on;
%plot showing the LV and FV intended tracks
subplot(2,1,1);
plot(LVIntendedTrack(:,2),LVIntendedTrack(:,1), 'r', FVIntendedTrack(:,2),
FVIntendedTrack(:,1), 'b');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('Leader', 'Follower', 2);
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV Intended (w/o DR Errors) Tracks Before and During the Data
Collection Process');
grid on;

%plot showing the LV and FV true tracks
```

```
subplot(2,1,2);
plot(LVTrueTrack(:,2),LVTrueTrack(:,1), 'r', FVTrueTrack(:,2),
FVTrueTrack(:,1), 'b');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('Leader', 'Follower', 2);
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV True (w/ DR Errors) Tracks Before and During the Data
Collection Process');
grid on;

%save the plots as both jpg and matlab fig files.  When the GUI
%is started the file numbers/names start at 1.  Figures are
%stored in %matlab_working_dir%\data\%fileName%
if(savePlots == 1)
    figFileName = strcat(figDirectory, num2str(figureNumber));
    saveas(gcf,figFileName, 'jpg');
    saveas(gcf,figFileName, 'fig');
    figureNumber = figureNumber + 1;
end

if(displayPlots ==0)   %if plots should be hidden after the save
    close;
end;
hold off;
```

3. GenerateSummaryTPPlot.m

```
function generateSummaryTPPlot(LVIntendedStartTPs, LVIntendedEndTPs,
LVIntendedTrack, LVTrueStartTPs, LVTrueEndTPs, LVTrueTrack, FVIntendedStartTPs,
FVIntendedEndTPs, FVIntendedTrack, FVTrueStartTPs, FVTrueEndTPs, FVTrueTrack,
NumOfTPs, figDirectory)
%generates a series of plots showing the LV and FV
%tracks during the correction process

global figureNumber;
global displayPlots;
global savePlots;

figure;
%plot showing the LV's true and intended test points
subplot(2,1,1);
hold on;
plot(LVIntendedTrack(2:((NumOfTPs*2)+1),2),LVIntendedTrack(2:((NumOfTPs*2)+1),1
),
'r',FVIntendedTrack(2:((NumOfTPs*2)+1),2),FVIntendedTrack(2:((NumOfTPs*2)+1),1)
, 'b', LVIntendedStartTPs(:,2), LVIntendedStartTPs(:,1),
'rO',LVIntendedEndTPs(:,2),LVIntendedEndTPs(:,1),'rX', FVIntendedStartTPs(:,2),
FVIntendedStartTPs(:,1),
'bO',FVIntendedEndTPs(:,2),FVIntendedEndTPs(:,1),'bX');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('Leader', 'Follower', 2);
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV Intended (w/o DR Errors) Tracks During the Data Collection
Process');

%add text to mark TP locations
for k=1:NumOfTPs
    countStart = strcat(' S',num2str(k));
    text(LVIntendedStartTPs(k,2), LVIntendedStartTPs(k,1),countStart);
    text(FVIntendedStartTPs(k,2), FVIntendedStartTPs(k,1),countStart);
```

```
end
grid on;

%plot showing the LV's true and intended test points
subplot(2,1,2);
plot(LVTrueTrack(2:((NumOfTPs*2)+1),2),LVTrueTrack(2:((NumOfTPs*2)+1),1),
'r',FVTrueTrack(2:((NumOfTPs*2)+1),2),FVTrueTrack(2:((NumOfTPs*2)+1),1), 'b',
LVTrueStartTPs(:,2), LVTrueStartTPs(:,1),
'rO',LVTrueEndTPs(:,2),LVTrueEndTPs(:,1),'rX', FVTrueStartTPs(:,2),
FVTrueStartTPs(:,1), 'bO',FVTrueEndTPs(:,2),FVTrueEndTPs(:,1),'bX');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('Leader', 'Follower', 2);
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV True (w/ DR Errors) Tracks During the Data Collection
Process');

%add text to mark TP locations
for k=1:NumOfTPs
    countStart = strcat(' S',num2str(k));
    text(LVTrueStartTPs(k,2), LVTrueStartTPs(k,1),countStart);
    text(FVTrueStartTPs(k,2), FVTrueStartTPs(k,1),countStart);
end
grid on;

%save the plots as both jpg and matlab fig files.  When the GUI
%is started the file numbers/names start at 1.  Figures are
%stored in %matlab_working_dir%\data\%fileName%
if(savePlots == 1)
    figFileName = strcat(figDirectory, num2str(figureNumber));
    saveas(gcf,figFileName, 'jpg');
    saveas(gcf,figFileName, 'fig');
    figureNumber = figureNumber + 1;
end

if(displayPlots ==0)   %if plots should be hidden after the save
    close;
end;
hold off;
```

4. GenerateSummaryResultsPlot.m

```
function generateSummaryResultsPlot(LVTrue, LVIntended, FVTrueUncorrected,
FVTrueCorrected, FVIntendedUncorrected, FVIntendedAfterRelCor, figDirectory)
%generates a series of plots showing the
%impact of the correction factor

global figureNumber;
global displayPlots;
global savePlots;


%first generate plots showing the relative positions before
%and after the correction
figure;
hold on;
%plot showing the LV true and FV uncorrected tracks
subplot(2,1,1);
plot(LVTrue(:,2),LVTrue(:,1), 'r', FVTrueUncorrected(:,2),
FVTrueUncorrected(:,1), 'b');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('LV True Pos', 'FV True w/o Cor', 2);
```

```
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV True Tracks Without Correction During Track Following (FV 100m
Behind LV');
grid on;

%plot showing the LV true and FV corrected tracks
subplot(2,1,2);
plot(LVTrue(:,2),LVTrue(:,1), 'r', FVTrueCorrected(:,2), FVTrueCorrected(:,1),
'b');
set(findobj(gca,'Type','line'), 'LineWidth', 1.5);
legend('LV True Pos', 'FV True with Cor', 2);
xlabel('East Postion (m)');
ylabel('North Position (m)');
title('LV and FV True Tracks with Correction During Track Following (FV 100m
Behind LV');
grid on;

%save the plots as both jpg and matlab fig files.  When the GUI
%is started the file numbers/names start at 1.  Figures are
%stored in %matlab_working_dir%\data\%fileName%
if(savePlots == 1)
    figFileName = strcat(figDirectory, num2str(figureNumber));
    saveas(gcf,figFileName, 'jpg');
    saveas(gcf,figFileName, 'fig');
    figureNumber = figureNumber + 1;
end

if(displayPlots ==0)   %if plots should be hidden after the save
    close;
end;
hold off;
```

M. MISCELLANOUS SUPPORT FUNCTIONS

1. Overview

This section provides the code for several miscellaneous functions, such as converting degrees to radians, calculating ranges, etc.

2. Range.m

```
function r = range(pointA, pointB)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% range - calculates the distance between two 3-D points
%
% r = the distance between pointA and pointB
% pointA, pointB - two 3-D points as [N,E,D].  The distance
% between these points will be returned
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r= sqrt(((pointA(1)-pointB(1))^2) + ((pointA(2)-pointB(2))^2) + ((pointA(3)-
pointB(3))^2));
```

3. CalculateRanges.m

```
function [ranges, errors] = calculateRanges(points1, points2, numOfPoints,
PRangingError, maxRangingError)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculateRanges - Calculates the ranges between 3-D points.
% ranges - numOfPoints x 1 matrix containing the ranges for
%          each of the points contained in points1 and points2
%          i.e. range between points1(1,:) and points2(1,:)
% points1 - one of two sets of 3-D points for which the range is
%           to be calculated (typically numOfPoints x 3)
% points2 - second of two sets of 3-D points for which the range is
%           to be calculated (typically numOfPoints x 3)
% numOfPoints - the number of points contained in each of
%               points1 and points2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ranges=[];  %start w/ empty matrix
errors = []; %start w/ empty matrix
for(k=1:numOfPoints)
    ranges = [ranges; range(points1(k,:), points2(k,:))];
    if(rand > (1-PRangingError))    %determine if a error should be included
        tempError = rand * maxRangingError;
        if(rand>0.5)                %random +/-
            ranges(k) = ranges(k) + tempError;
            errors = [errors;tempError];
        else
            ranges(k) = ranges(k) - tempError;
            errors = [errors; -tempError];
        end
    else
        errors = [errors; 0];
    end
end
```

4. RotateRefFrame.m

```
function newPoints = rotateRefFrame(angle, points, numberOfPoints)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rotateRefFrame - Rotates the reference frame for a set of points
% newPoints = numberOfPoints x 3 matrix containing the rotated points
%              points are returned in a n-e-d format/order
% angle = then angle to rotate the points in degrees where 0degrees
%         is north
% points = a numberOfPoints x 3 matrix containing the points to be
%          rotated.  The order should be north, east, down.
% numberOfPoints = the number of points contained in "points"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rotAngle = deg2rad(angle);  %the angle of rotation in rads

newPoints=[];   %start with an empty matrix so can be used in loop

for(k=1:numberOfPoints)
    newPoints = [newPoints; points(k,1), points(k,2), points(k,3), 1];
        %setup a matrix containing the test points to be used for rotation
end

multFactor = [cos(rotAngle),sin(rotAngle),0,0;-sin(rotAngle), cos(rotAngle), 0,
0; 0,0,1,0;0,0,0,1];
    %setup the transformation matrix

newPoints = newPoints * multFactor;
    %rotate the points

newPoints = newPoints(:,1:3);
    %truncate the extra 1's in the right most column
```

5. Deg2rad.m

```
function rads = deg2rad(degs)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% deg2rad - Converts degrees to radians.  It also corrects
% for degs <0 and deg>=360.
%
% rads = the radian value of degs (between 0 and 2pi)
% degs = the value to be converted (in degrees).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rads = degs * (pi/180);
while(rads>=(2*pi))
    rads = rads - (2*pi);
end
while(rads<0)
    rads = rads + (2*pi);
end
```

6. Rad2deg.m

```
function degs = rad2deg(rads)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rad2deg - Converts radians to degrees.  It also corrects
% for degs <0 and deg>=360.
%
% degs = the angle after conversion to degrees (between 0 and 359.99)
% rads = the radian angle to be converted to degrees.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

degs = rads * (180/pi);
while(degs >= 360)
    degs = degs - 360;
end;
while(degs <0)
    degs = degs + 360;
end;
```

7. DetermineTPConfigForParallel.m

```
function configuration = determineTPConfigForParallel(testPoints)
% Determines the configuration of the test points in the
% event the user wishes to use the parallel correction
% method.
% Returns 1 if TP2 (origin for the correction frame) is NE of the
%   target vehicle
% Returns 2 if TP2 (origin for the correction frame) is SE of the
%   target vehicle
% Returns 3 if TP2 (origin for the correction frame) is NW of the
%   target vehicle
% Returns 4 if TP2 (origin for the correction frame) is SW of the
%   target vehicle

configuration = 0;  %set with default of zero in case of failure

if(testPoints(1,2) > 0 & testPoints(1,1) < testPoints(2,1))
    configuration = 1;  %TP2 is NE of the TV
end

if(testPoints(1,2) > 0 & testPoints(1,1) > testPoints(2,1))
    configuration = 2;  %TP2 is SE of the LV
end
```

```
if(testPoints(1,2) < 0 & testPoints(1,1) < testPoints(2,1))
    configuration = 3;  %TP2 is NW of the LV
end

if(testPoints(1,2) < 0 & testPoints(1,1) > testPoints(2,1))
    configuration = 4;  %TP2 is SW of the LV
end
```

APPENDIX C. AUV WORKBENCH CODE

A. INTRODUCTION

This section contains selected portions of the AUV Workbench code that provide details regarding the implementation of the follow-the-leader algorithm and the modifications completed to support the simulation of cooperative behaviors.

B. SAMPLE VRML FILE

1. Overview

This section provides an example VRML that is generated by the workbench and rendered in the Xj3D browser window. This VRML file invokes the primary workbench java classes and it accepts data from the executing Java code to manipulate the position and orientation of the vehicles in the virtual world.

2. GeneralAUVMissionScenerio.wrl

```
#VRML V2.0 utf8
# X3D-to-VRML-97 XSL translation autogenerated by X3dToVrml97.xsl
# http://www.web3D.org/TaskGroups/x3d/translation/X3dToVrml97.xsl

# [X3D] VRML V3.0 utf8

# [head]
# [meta] filename: GeneralAUVMissionScenario.x3d
# [meta] description: General Scene for AUV Mission Visualization Workbench
# [meta] author: Doug Horner
# [meta] created: 020203
# [meta] revised: 020203
# [meta] revised: 28JUN03 changed viewpoints to resolve j3d error
# [meta] version: 0.1
# [meta] generator: X3D-Edit,
http://www.web3d.org/TaskGroups/x3d/translation/README.X3D-Edit.html
# [Scene]

NavigationInfo {
  type [ "EXAMINE" "ANY" ]
}
DEF GlobalClock TimeSensor {
  loop TRUE
}
Background {
  topUrl [ "urn:web3d:media:textures/panoramas/ocean_3_top.jpg"
"../environment/ocean_3_top.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_top.jpg"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_top.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_top.jpg"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_top.jpg"
"http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_top.jpg"
```

```
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_top.jp
g" ]
  skyColor [ 0.4 0.4 0.1, 0.4 0.4 0.1, 0 0.1 0.3, 0 0.2 0.6, 0.8 0.8 0.8  ]
  skyAngle [ 0.1, 0.15, 1.309, 1.571 ]
  rightUrl [ "urn:web3d:media:textures/panoramas/ocean_3_right.jpg"
"../environment/ocean_3_right.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_right.jpg"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_right.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_right.jpg"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_right.jp
g" "http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_right.jpg"
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_right.
jpg" ]
  leftUrl [ "urn:web3d:media:textures/panoramas/ocean_3_left.jpg"
"../environment/ocean_3_left.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_left.jpg"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_left.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_left.jpg"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_left.jpg
" "http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_left.jpg"
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_left.j
pg" ]
  groundColor [ 0 0 0, 0 0.1 0.3, 0 0.2 0.5, 0 0.3 0.8 ]
  groundAngle [ 0.1, 1.309, 1.571  ]
  frontUrl [ "urn:web3d:media:textures/panoramas/ocean_3_front.jpg"
"../environment/ocean_3_front.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_front.jpg"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_front.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_front.jpg"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_front.jp
g" "http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_front.jpg"
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_front.
jpg" ]
  bottomUrl [ "urn:web3d:media:textures/panoramas/ocean_3_bottom.jpg"
"../environment/ocean_3_bottom.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_bottom.jpg
"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_bottom.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_bottom.jpg
"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_bottom.j
pg" "http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_bottom.jpg"
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_bottom
.jpg" ]
  backUrl [ "urn:web3d:media:textures/panoramas/ocean_3_back.jpg"
"../environment/ocean_3_back.jpg"
"http://www.web3d.org/WorkingGroups/media/textures/panoramas/ocean_3_back.jpg"
"http://www.web3D.org/TaskGroups/x3d/translation/examples/UniversalMediaPanoram
as/ocean_3_back.jpg"
"http://www.web3dmedia.com/UniversalMedia/textures/panoramas/ocean_3_back.jpg"
"http://www.officetowers.com/UniversalMedia/textures/panoramas/ocean_3_back.jpg
" "http://geometrek.com/UniversalMedia/textures/panoramas/ocean_3_back.jpg"
"http://www.sc.ehu.es/ccwgamoa/UniversalMedia/textures/panoramas/ocean_3_back.j
pg" ]
}
Group {
  children [
      Shape {
        appearance Appearance {
          material Material {
            emissiveColor 0 .5 1
          }
        }
        geometry IndexedLineSet {
          coordIndex [ 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20 ]
          coord Coordinate {
            point [ 10  -5 10,   10  -5 210,  25  -5 210,
                    25  -5 10,   40  -5 10,   40  -5 210,
                    55  -5 210,  55  -5 10,   70  -5 10,
                    70  -5 210,  85  -5 210,  85  -5 10,
                    100  -5 10,  100  -5 210, 115  -5 210,
                    115  -5 10,  130  -5 10,  130  -5 210,
                    145  -5 210, 145  -5 10,  -50  -5 10 ]
          }
        }
      }
      Viewpoint {
        position 25 800 100
        orientation 1 0 0 -1.57
        description "800M above AUV"
      }
      Viewpoint {
        position 25 200 100
        orientation 1 0 0 -1.57
        description "200M above AUV"
      }
    ]
}
DEF AriesTransform Transform {
  children [
      Inline {
        url [ "../VrmlFiles/vehicles/Aries/AriesPrototype.wrl" ]
      }
 ### Hint:  For maximum portability, append alternate "http://address" to
url='"../VrmlFiles/vehicles/Aries/AriesPrototype.wrl"'
      Viewpoint {
        position 1.0 100.0 0.0
        orientation 1.0 0.0 0.0 -1.57
        fieldOfView 1.1
        description "100M above ARIES"
      }
      Viewpoint {
        position 1.0 10.0 0.0
        orientation 1.0 0.0 0.0 -1.57
        fieldOfView 1.1
        description "ARIES Top View"
      }
      Viewpoint {
        position 10.0 0.0 0.0
        orientation 0.0 1.0 0.0 1.57
        fieldOfView 1.1
        description "ARIES Front View"
      }
      Viewpoint {
        position -15 10 0
        orientation 1 0 0 -.85
        description "ARIES Aft looking forward"
      }
    ]
}
DEF RemusTransform Transform {
  children [
      Inline {
        url [ "../VrmlFiles/vehicles/Remus.wrl" ]
      }
 ### Hint:  For maximum portability, append alternate "http://address" to
url='"../VrmlFiles/vehicles/Remus.wrl"'
      Viewpoint {
        position 1.0 100.0 0.0
        orientation 1.0 0.0 0.0 -1.57
        fieldOfView 1.1
        description "100M above REMUS"
      }
      Viewpoint {
        position 1.0 10.0 0.0
        orientation 1.0 0.0 0.0 -1.57
        fieldOfView 1.1
        description "REMUS Top View"
      }
      Viewpoint {
        position 10.0 0.0 0.0
        orientation 0.0 1.0 0.0 1.57
        fieldOfView 1.1
        description "REMUS Front View"
      }
      Viewpoint {
        position -15 10 0
        orientation 1 0 0 -.85
        description "REMUS Aft looking forward"
      }
   ]
}
DEF ScenarioControllerScript Script {
  eventOut     SFVec3f     RemusTranslation
  eventOut     SFRotation  RemusOrientation
  eventOut     SFVec3f     AriesTranslation
  eventOut     SFRotation  AriesOrientation
  eventIn      SFFloat     TimeFractionChanged
  mustEvaluate TRUE
  directOutput FALSE
url [ "AUVVRMLController.class"
 ]
}
 ### Hint:  For maximum portability, append alternate "http://address" to
url='"AUVVRMLController.class"'
ROUTE GlobalClock.fraction_changed TO
ScenarioControllerScript.TimeFractionChanged
ROUTE ScenarioControllerScript.RemusTranslation TO
RemusTransform.set_translation
ROUTE ScenarioControllerScript.RemusOrientation TO RemusTransform.set_rotation
ROUTE ScenarioControllerScript.AriesTranslation TO
AriesTransform.set_translation
ROUTE ScenarioControllerScript.AriesOrientation TO AriesTransform.set_rotation
```

C. JAVA / VRML INTERFACE

1. Overview

The code in this section is invoked by the VRML file provided in the previous section and it serves as the interface between the Java-based vehicle models and the VRML virtual world.

2. AUVVRMLController.java

```
/*
 * AUVVRMLController.java is a script node interface that updates the position
 * and orientation of multiple AUVs in a VRML scene.  This class is based on
the
 * EntityController class originally developed by Ekrem Serin.
 * Current Version Author: Dan Kucik
 *
 */

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.util.*;


/**
 * AUVVRMLController.java is a script node interface that updates the position
 * and orientation of multiple AUVs in a VRML scene.  This class is based on
the
 * EntityController class developed by Ekrem Serin (modified by James Harney).
 * All vehicles are controlled from a single node (one instance of the
 * AUVVRMLController class) so that all dyanmics/behavior threads are created
 * within a single instance.  This approach was taken to allow for simple
 * vehicle-to-vehicle communications and for data collection reasons.
 *
 * @author Ekrem Serin
 * @author James Harney
 * @author Dan Kucik
 * @version 01AUG03
 *
 */
public class AUVVRMLController extends Script
{

   private SFFloat fractionChanged = null;
      //used to trigger position/orientation updates in the scene.
      //to contain the fraction_changed value of a TimeSensor in the scene

   private SFVec3f remusPosition = null;
      //holds the position of the lead AUV

   private SFRotation remusOrientation = null;
      //holds the leader's orientation

   private SFVec3f ariesPosition = null;
      //holds the position of the following AUV

   private SFRotation ariesOrientation = null;
      //holds the follower's orientation


      //varibles to hold position and orientation information obtained from
      //the dynamics models until a time event occurs, which will then feed
      //these values to the scene.
   private float remusXPosition = 0;
   private float remusYPosition = 0;
   private float remusZPosition = 0;
   private float remusHeading = 0;
   private float ariesXPosition = 0;
   private float ariesYPosition = 0;
   private float ariesZPosition = 0;
   private float ariesHeading = 0;

   public RemusDynamics remusDynamics;
      //will point to an instance of the leader's dynamics class

       public Thread remusDynamicsThread;
      //thread used to execute the leader's dynamics class

   public AriesDynamics ariesDynamics;
      //points to an instance of the dynamics model for the following vehicle

   public Thread ariesDynamicsThread;
      //thread used for execution of the follower's dynamics

   public ReportGeneration reporting;
      //pointer to an instance of the ReportGeneration class used to generate
      //and output status reports to a file.

   public Thread reportingThread;
      //thread for execution of the reporting code

   private boolean ariesUpdateComplete = false;
   private boolean remusUpdateComplete = false;
   private int numberOfMissedCycles = 0;
      //flags used to ensure vehicles are updated at same rate to prevent
      //"jittery" appearance.

   /**
    * Constructor does nothing since the class is envoked by the VRML scene
    */
   public AUVVRMLController()
    {
       //do nothing
    }

   /**
    * Initializes the handles needed to access the scene, creates instances
    * of the dynamics models, and starts the threads to run the dynamics
classes.
    *
    * @param none
    * @return void
    */
    public void initialize()
    {
      //get the handles for the VRML fields
      fractionChanged = (SFFloat) getField("TimeFractionChanged");
      remusPosition = (SFVec3f) getField("RemusTranslation");
      remusOrientation = (SFRotation)getField("RemusOrientation");
      ariesPosition = (SFVec3f) getField("AriesTranslation");
      ariesOrientation = (SFRotation)getField("AriesOrientation");

      //get the initial position and orientation of the AUVs in the scene
      remusXPosition = remusPosition.getX();
      remusYPosition = remusPosition.getY();
      remusZPosition = remusPosition.getZ();
      remusHeading = 0f;
      ariesXPosition = ariesPosition.getX();
      ariesYPosition = ariesPosition.getY();
      ariesZPosition = ariesPosition.getZ();
      ariesHeading = 0f;
      remusDynamics = new RemusDynamics (this);
         //create an instace of the leader's dynamics class

      remusDynamicsThread = new Thread (remusDynamics);
         //create a thread to run the leader's dynamics class

      remusDynamicsThread.start();
         //start the leader's dyanmics thread


      //delay while Remus reads XML files
      boolean setupDone = false;
      while(setupDone == false)
      {  //would lock on empty loop, so broke-up
         setupDone = remusDynamics.isStartupComplete();
      }

      ariesDynamics = new AriesDynamics(this);
         //create an instance of the dynamics for the follower vehicle

      ariesDynamicsThread = new Thread(ariesDynamics);
         //create a thread for execution of the follower dynamics

      ariesDynamicsThread.start();
         //start the followers's dyanmics thread


      reporting = new ReportGeneration(remusDynamics, ariesDynamics);
         //create an instance of the report generation class which is used to
         //output status reports to a file for post-processing

      reportingThread = new Thread(reporting);
         //run reporting class in its own thread

      reportingThread.start();
         //start the reporting dyanmics thread


    }

    /**
     * Updates the position and orientation of both REMUS and ARIES.  The
     * update is triggered by the global clock in the VRML scene.
     *
     * @param  evt - the timeSensor event from the VRML scene
     * @return void
     */
   public void processEvent(vrml.Event evt)
   {
      if((ariesUpdateComplete && remusUpdateComplete) || numberOfMissedCycles
>10)
      {
         remusPosition.setValue(remusXPosition, remusYPosition,
remusZPosition);
         remusOrientation.setValue(0f, -1f, 0f, remusHeading);
         ariesPosition.setValue(ariesXPosition, ariesYPosition,
ariesZPosition);
         ariesOrientation.setValue(0f, -1f, 0f, ariesHeading);
         ariesUpdateComplete = false;
         remusUpdateComplete = false;
         numberOfMissedCycles = 0;
      }
      else
      {
         numberOfMissedCycles++;
      }
   }


    /**
     * Updates the position and orientation for the ARIES AUV
     *
     * @param  x - ARIES' x position (meters)
     * @param  y - ARIES' y position (meters)
     * @param  z - ARIES' z position (meters)
     * @param  heading - ARIES' current heading (rads)
     * @return void
     */
   public void updateAries(float x, float y, float z, float heading)
   {
      ariesXPosition = x;
      ariesYPosition = y;
      ariesZPosition = z;
      ariesHeading = heading;
      ariesUpdateComplete = true;
   }


    /**
     * Updates the position and orientation for the REMUS AUV
     *
     * @param  x - REMUS' x position (meters)
     * @param  y - REMUS' y position (meters)
     * @param  z - REMUS' z position (meters)
     * @param  heading - REMUS' current heading (rads)
     * @return void
     */
   public void updateRemus(float x, float y, float z, float heading)
   {
      remusXPosition = x;
      remusYPosition = y;
      remusZPosition = z;
      remusHeading = heading;
      remusUpdateComplete = true;
   }

    /**
     * Called on shutdown
     * Closes all open files and destroys the threads
     */
   public void shutdown()
   {
      System.out.println("Shutdown called");
      ariesDynamicsThread.destroy();
      remusDynamicsThread.destroy();
      ReportGeneration.quit();
      reportingThread.destroy();
   }



   /**
    * Gets the pointer to the REMUS dynamics instance.
    *
    * @params - none
```

```
     * @return pointer to the REMUS dynamics instance
     */
    public RemusDynamics getPointerToRemus()
    {
       return remusDynamics;
    }

    /**
     * Gets the pointer to ARIES' dynamics instance.
     *
     * @params - none
     * @return pointer to the Aries dynamics instance
     */
    public AriesDynamics getPointerToAries()
    {
       return ariesDynamics;
    }

    /**
     * Gets the pointer to report generation instance.
     *
     * @params - none
     * @return pointer to the report generation instance
     */
    public ReportGeneration getPointerToReportGeneration()
    {
       return reporting;
    }

}//end AUVVRMLController class
```

D. ARIES DYNAMICS AND IMPLEMENTATION OF “FOLLOW-THE-LEADER”

1. Overview

This section contains the code defining ARIES’ dynamics and the implementation of the follow-the-leader algorithm.

2. AriesDynamics.java

```
import Jama.*;
import java.util.*;

import org.jdom.*;
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.transform.*;

import java.io.*;
import java.net.*;
import java.util.*;

import vrml.*;
import vrml.field.*;
import vrml.node.*;
import mil.navy.nps.dis.*;

/**
 * AriesDynamics.java
 * The 6 DOF dyanmics model for the Aries AUV.
 *
 * Version History
 * Code originally written by Dave Marco in C
 * Converted over to Java by Doug Horner 1/26/02
 * Modified to implement "follow the leader" by Dan Kucik 5/03-9/03
 *
 * @author Dave Marco
 * @author Doug Horner
 * @author Dan Kucik
 * @version 02AUG03
 */
public class AriesDynamics implements Runnable
{
   AUVVRMLController execSc;      //Executor Script for the VRML scene

   final double DEFAULT_SPEED = 1.543;

   //The following variables contain the data received from the LV via ACOMMS
   //These variables are intended for use by getNextWaypoint().
   double LVHeading = 0;   //LV current heading
   double LVCourse = 0;
   double LVXPosition = 0;
   double LVYPosition = 0;
   double LVZPosition = 0;
   double LVSpeed = 0;
   double LVCurrentWPX = 0;
   double LVCurrentWPY = 0;
   double LVCurrentWPZ = 0;
   double LVNextWPX = 0;
   double LVNextWPY = 0;
   double LVNextWPZ = 0;
   boolean LVMissionComplete = false;
   double carrot = 8;
      //distance between LV and FV (m)

   //the following vars are for controlling the loiter behavior
   boolean currentlyLoitering = false;
   double loiterCenterX = 0;
   double loiterCenterY = 0;
   int loiterPtCount = 0;
   boolean LVWPReached = true;
   int LVWaypointCount = 0;

   double XWay = 0;     //waypoint currently being processed
   double YWay = 0;

   double PrevXWay = 0; //previously processed waypoint
   double PrevYWay = 0;

   double X = 0;        //current position of the vehicle
   double Y = 0;

   //start of dynamics coefs
   double Iy = 3.45; //kg/m^3
   double Iz = 3.45;


   double L = 1.33; //length in meters
   double W = 299; //Weight in N
   double g = 9.81; //Acceleration of gravity in m/s^2
   double m = W/g; //Mass in kilograms
   double V = 1.543; //Max Speed in m/s
   double rho = 1030; //Density of Salt H20 in kg/m^3
   double D = .191; //Max diameter in meters

   //State Model Parameters
   double U = 1.543;
   double Boy = 299;
   double xg = 0;
   double yg = 0;
   double zg = .0196;

   double Nvdot = 1.93;
   double Nrdot = -4.88;
   double Yvdot = -35.5;
   double Yrdot = 1.93;
   double Nv = -4.47;
   double Nr = -6.87;
   double Yv = -66.6;
   double Yr = 2.2;
   double Nd = -9.8857;
   double Yd = 14.4571;

   double currentX, currentY, currentZ, heading;     //floats for the
updateEntity method in EntityController.java

   double RadCurv;
   double SideSlip;
   Matrix xss;

  /**
    * Constructor for the ARIES dynamics model
    *
    * @param   exS - Handle for the VRML scene controller
    */
   public AriesDynamics(AUVVRMLController exS)
   {
      execSc = exS;
      setupInsert();

   } //End of Contstuctor for AUV class


   public void run()
   {
      int timeToCompleteRun = 1800;
      boolean UnderseaNetworkExists = false;

      double comp = m - Yvdot;
      double comp1 = Iz - Nrdot;
      int maxTime = 10000;
      int maxCourseLegs = 100;

      double[][] vals = {{comp, -Yrdot, 0},{-Nvdot, comp1, 0},{0, 0, 1}};
      Matrix MM = new Matrix(vals);

      double comp2 = Yr - m*V;
      double[][] vals1 = {{Yv, comp2, 0},{Nv, Nr, 0},{0, 1, 0}};
      Matrix AA = new Matrix(vals1);

      double[][] vals2 = {{Yd}, {Nd}, {0}};
      Matrix BB = new Matrix(vals2);
      System.out.println("The Matrix BB is ");
      BB.print(3,3);

      Matrix A = MM.inverse().times(AA);
      System.out.println("The Matrix A is ");
      A.print(3,3);

      Matrix B = MM.inverse().times(BB);
      System.out.println("The Matrix B is ");
      B.print(3,3);

      double[][] vals4 = {{0}, {0}, {1}};
      Matrix C = new Matrix(vals4);

      Matrix A2 = A.getMatrix(0,1,0,1);
      System.out.println("The Matrix A2 is ");
      A2.print(3,3);

      Matrix B2 = B.getMatrix(0,1,0,0);

      xss = A2.inverse().times(B2);

      EigenvalueDecomposition E = A2.eig();
      E.getV();

      double RadGy = StrictMath.sqrt(Iz/(W/g)); //in meters

      RadCurv = U/(xss.get(0,0));

      double Pi = StrictMath.PI;
      SideSlip = StrictMath.atan2(xss.get(0,0),U)*180/Pi;

      double[][] vals5 = {{.7690, -.6000, 0} };
      Matrix k = new Matrix(vals5);
      System.out.println("The Matrix k is ");
      k.print(3,3);

      Matrix Ac = A.minus(B.times(k));
      System.out.println("The Matrix Ac is ");
      Ac.print(3,3);

      double  dt = 0.125/2; //incremental time changes

      int time = (int) (timeToCompleteRun * (1 / dt));
      System.out.println("Total Array Size is equal to " + time);

      double[] t = new double[time];

      for (int n = 0; n< time; n++)
      {
         t[n] = n * .125/2;
      }

      double DegRad = Pi/180;
      double RadDeg = 180/Pi;

      //Set initial Conditions
      int start = 10;
      double[] v = new double[time];          //This used to be an array of
size maxTime check to see why
      v[0] = 0.0;
      double[] r = new double[time];
      r[0] = 0.0;
      double[] rRM = new double[time];
      rRM[0] = 0.0;

      double[] psi = new double[time];

       //This is the Initial Heading of the Vehicle
      psi[0] = 50 * DegRad;

       //Read in the track/course XML files here

      int NumberOfTracks = 7;

              //This is the Initial Position of the Vehicle

      double SurfaceTime = 0;
      double SurfPhase = 0;

      double rCom = 0.0;
      double WR = 2.0;
      double rabbit = 9; //Represents the distance between ARIES and the rabbit

      double a = 8.75;
      double b = -13.5;

      double[][] xArray = new double[3][time];
      Matrix x = new Matrix(xArray);
      x.set(0, 1, v[0]);
      x.set(1, 1, r[0]);
      x.set(2, 1, psi[0]);

      double Lam1 = 2.0;
      double Lam2 = 1.0;

      double EtaFlightHeading = 0.5; //Lowered this from 1.0 on ARIES model
      double PhiFlightHeading = 0.1; //Lowered this from .5 on ARIES model

       //Below for tanh
      double EtaCTE = 0.05; //Play with to fine tune
      double EtaCTEMin = 1.0;
      double PhiCTE = 0.2;

       //Calculating the segment lengths between each waypoint
       //The first leg is initialized outside the the for loop
       // because it is based on the initial starting point
      int currentWP = 1;
      double[] tempWP = getNextWaypoint(0, currentWP);
      XWay = tempWP[0];
      YWay = tempWP[1];

      double Xdist = StrictMath.pow((XWay - PrevXWay),2);
      double Ydist = StrictMath.pow((YWay - PrevYWay),2);

      double segLen = StrictMath.sqrt(Xdist + Ydist);

      double psiTrack = StrictMath.atan2(YWay - PrevYWay, XWay - PrevXWay);

      int j =0;     //Originally set to 1 in MATLAB but switched to 0 because
java arrays begin with 0 instead of 1

      double[] Sigma = new double[time]; //What is Sigma for?
      double[] DepthCom = new double[time]; //Depth Command variable
      double[] dr = new double[time]; //rudder commands at each time step t
      double[] drl = new double[time]; //lagged rudder;

      drl[0] = 0.0;
      DepthCom[0] = 5.0; //depth command in meters

      double WayPointVertDistCom = 5.0;

      boolean SurfaceTimerActive = false;

      double[] XWayError = new double[time];
      double[] YWayError = new double[time];
      double[] psiCont = new double[time];
      double[] psiErrorCTE = new double[time];
      double[] s = new double[time];

      double[][] xDotArray = new double[3][time];
      Matrix xDot = new Matrix(xDotArray);

      xDot.set(0,1,0);
      xDot.set(1,1,0);
      xDot.set(2,1,0);

      int sign = 0;
      double Ratio;

      double[] ss = new double[time];
      double[] dp = new double[time];
      double[] cte = new double[time];
      double[] LOS = new double[time];
      double[] psiComLOS = new double[time];
      double[] SigmaFlightHeading = new double[time];

      double[] psiErrorLOS = new double[time];

      double surfaceWait = 0;                 //I've initialized to 0 without
checking - check!
      boolean surfaceTimerActive = false;
      double cc;
      boolean missionExecuting = true;

      double[] posit = new double[181];

      int deg = -90;             //Starting point for constructing an array
from -90 to 90 for OA

      for (int i = 0; i < 181; i++)
      {
         posit[i] = deg;
         deg++;
      }

      double[] w1 = new double[time];
      double[] w2 = new double[time];

      int cntr = 0; //Temporary variable for debugging


      while(missionExecuting)
      {
           //Beginning of the CTE controller
         for (int i = 0; i <= (time - 1); i++) //Can abbreviate the for loop
for testing purposes
```

```
         {
            DepthCom[i] = WayPointVertDistCom;

            XWayError[i] = XWay - X;
            YWayError[i] = YWay - Y;

            //Dewrap psi to within +/- 2.0 * Pi
            psiCont[i] = psi[i];

            while(StrictMath.abs(psiCont[i]) > 2.0 * Pi)
            {
               psiCont[i] = psiCont[i] - signum(psiCont[i]) * 2.0 * Pi;

            }

            psiErrorCTE[i] = psiCont[i] - psiTrack;

             //Dewrap psi_error to within +/- pi;

            while (StrictMath.abs(psiErrorCTE[i]) > Pi)
            {
               psiErrorCTE[i] = psiErrorCTE[i] - signum(psiErrorCTE[i]) * 2.0 *
Pi;
            }

            double Beta = 0.0;
            double cpsiE = StrictMath.cos(psiErrorCTE[i] + Beta);
            double spsiE = StrictMath.sin(psiErrorCTE[i] + Beta);

            s[i] = (XWayError[i] * (XWay - PrevXWay)) + (YWayError[i] * (YWay -
PrevYWay)) ;

            // s is distance to go projected to track line(goes from 0-100%L)
            s[i] = s[i] / segLen;
            Ratio = (1.0 - s[i]/segLen) * 100;

            //Radial distance to go to next Waypoint
            ss[i] = StrictMath.sqrt(StrictMath.pow(XWayError[i],2) +
StrictMath.pow(YWayError[i],2));

            dp[i] = StrictMath.atan2((YWay - PrevYWay) , (XWay - PrevXWay)) -
StrictMath.atan2( YWayError[i], XWayError[i]);

            if(dp[i] > Pi)
            {
               dp[i] = dp[i] - 2.0 * Pi;
            }

            cte[i] = s[i]*StrictMath.sin(dp[i]);

            if(StrictMath.abs(psiErrorCTE[i]) >= 0.0 * Pi/180.0)   // || s[i] <
0.0 ) used to read 40.0*Pi not 00.0*pi
            {
               //Use LOS Control
               LOS[i] = 1;
               psiComLOS[i] = StrictMath.atan2(YWayError[i] , XWayError[i]);

               // Construct Bearing/Range to each obstacle
               cc = 0;
               boolean increaseWeight = false;

               psiErrorLOS[i] = psiTrack - psiCont[i] -
StrictMath.atan2(cte[i], rabbit);
```

```
               if (StrictMath.abs(psiErrorLOS[i]) > Pi)
               {
                  psiErrorLOS[i] = psiErrorLOS[i] - 2.0 * Pi *
psiErrorLOS[i]/StrictMath.abs(psiErrorLOS[i]);
               }

               SigmaFlightHeading[i] = ((-.0442 * v[i]) * 0.0 + .5004 * (rCom -
r[i]) + .8647 * psiErrorLOS[i]);

               double u = SigmaFlightHeading[i] / PhiFlightHeading;

               double calTanh = (StrictMath.exp(u) - StrictMath.exp(-u)) /
(StrictMath.exp(u) + StrictMath.exp(-u));

               dr[i] = (-k.get(0,0) * v[i]) * 0.0 -k.get(0,1) * r[i] -
EtaFlightHeading * calTanh;
            }

            else
            {
               // Use CTE Controller
               LOS[i] = 0;

               if (cpsiE != 0)
                                   //Trap Div. by zero !
               {
                  //SMC Soln
                  Sigma[i] = U * rRM[i] * cpsiE + Lam1 * U * spsiE + Lam2 *
cte[i];
                  dr[i] = (1.0/(U * a * cpsiE)) * (-U * b * rRM[i] * cpsiE +
StrictMath.pow(U*rRM[i], 2) * spsiE
                     - Lam1 * U * rRM[i] * cpsiE - Lam2 * U * spsiE - 2.0 *
EtaCTE * (Sigma[i]/PhiCTE));
               }
               else
               {
                  dr[i] = dr[i-1];
               }
            }//end of CTE Controller

            //Surface Phase Logic (Independent of LOS or CTE)
            if (SurfPhase == 1)  //Check on this assignment
            {
               if (surfaceTimerActive = false)
               {
                  if (Ratio > 40.0)
                  {
                     //Start a timer
                     surfaceTimerActive = true;
                     DepthCom[i] = 0.0;
                     surfaceWait = SurfaceTime + t[i];
                  }
               }
            }

            if (surfaceTimerActive == true)
            {
               if (t[i] >= surfaceWait)
               {
                  surfaceTimerActive = false;
                  DepthCom[i] = WayPointVertDistCom;
                  SurfPhase = 0;
               }
               else
               {
                  DepthCom[i] = 0;
               }
            }

            if (StrictMath.abs(dr[i]) > rabbit * Pi/180 )   //change from 0.4
            {
               dr[i] = rabbit * Pi/180 * signum(dr[i]);     //simulates follow
the rabbit
            }

            drl[i] = 0.4 * signum(drl[i]);

         //initialized the xDot Matrix outside the CTE for loop

            xDot.set(0, i + 1, (A.get(0,0) * v[i] + A.get(0,1) * r[i] +
B.get(0,0) * dr[i]));      //Probably a more elegant way of writing
            xDot.set(1, i + 1, (A.get(1,0) * v[i] + A.get(1,1) * r[i] +
B.get(1,0) * dr[i]));     //this in Jama but it'll do for a start
            xDot.set(2, i + 1, r[i]);

            x.set(0, i+1, x.get(0,i) + dt * xDot.get(0, i));
            x.set(1, i+1, x.get(1,i) + dt * xDot.get(1, i));
            x.set(2, i+1, x.get(2,i) + dt * xDot.get(2, i));

            v[i+1] = x.get(0, i+1);
            r[i+1] = x.get(1, i+1);
            psi[i+1] = x.get(2, i+1);
            rRM[i+1] = r[i+1];

         //Wave Motions
            double Uc = 0.0;
            double Vc = 0.0;

        //Kinematics

            X = X + (Uc + (U)*StrictMath.cos(psi[i]) - v[i] *
StrictMath.sin(psi[i]) )* dt;

            Y = Y + (Vc + (U)*StrictMath.sin(psi[i]) - v[i] *
StrictMath.cos(psi[i]) )* dt;

        //note these are in the VRML frame
            currentX = X;
            currentY = DepthCom[i];
            currentZ = Y;

            heading = psiCont[i];

            int modulus = i % 2;

            if (modulus == 0)
            {
            //Using the modulus operator for base 16 so that if equal to zero
            //it reports to the EntityController the present position
            //intention is not to overwhelm the visualization with incremental
position updates
            //mod16 choosen because want once a second updates and this program
updates positions
            //16 times per second (i.e. dt = .125/16)
```

```
               execSc.updateAries((float)currentX, (float)currentY,
(float)currentZ, (float)heading);
            }

            //Setting thread to sleep for 62ms to slow down the execution of
the processor
            //So that updates to the AUV should approximate real time
            //62ms * 16 ~= 1000ms = 1 sec

            try
            {
               Thread.sleep(62);
            }
            catch(Exception e)
            {
               e.printStackTrace();
            }

      //Check to see if we are within the Watch_Radius or if we passed the wypt
      //Change to next wypt if radial distance to go is less than rabbit
distance
      //or if we passed the wypt or if we are within the Watch_Radius

            if (StrictMath.sqrt(StrictMath.pow(XWayError[i], 2) +
StrictMath.pow(YWayError[i], 2)) <= WR || s[i] < 0.0 || ss[i] < rabbit)
            {
               System.out.println("Waypoint reached");

               PrevXWay = XWay;
               PrevYWay = YWay;
               currentWP++;
               tempWP = getNextWaypoint(0, currentWP);
               XWay = tempWP[0];
               YWay = tempWP[1];
               WayPointVertDistCom = tempWP[2];
               Xdist = StrictMath.pow((XWay - PrevXWay),2);
               Ydist = StrictMath.pow((YWay - PrevYWay),2);

               segLen = StrictMath.sqrt(Xdist + Ydist);

               psiTrack = StrictMath.atan2(YWay - PrevYWay, XWay - PrevXWay);

               if (LVMissionComplete)
               {
                  System.out.println("ARIES Mission Completed");
                  missionExecuting = false;          //boolean variable for
terminating the infinite loop
                  break;
               }
               else
               {
                  System.out.println("ARIES numberOfTrack is equal to: " + j);
                  j++;
               }
            }

            dr[i+1] = dr[i];
            cte[i + 1] = cte[i];
            s[i+1] = s[i];
            ss[i+1] = ss[i];

         } //End of for loop
      } //End of while loop
   } //End of run method

       /**
        * Collapse number down to +1 0 or -1 depending on sign.
        * Typically used in compare routines to collapse a difference
        * of two longs to an int.
        *
        * @param diff usually represents the difference of two long.
        *
        * @return signum of diff, +1, 0 or -1.
        **/

   private static int signum(double diff)
   {
      if ( diff > 0 )
      {
         return 1;
      }
      if ( diff < 0 )
      {
         return -1;
      }
      else
         return 0;
   } // end signum


   /**
    * Determines if the vehicle is currently processing its last waypoint.
    *
    * @param   none
    * @return  flag indicating if the vehicle is currently processing its last
    *          waypoint (true if last waypoint)
    */
   public boolean isLastWaypoint()
   {
         return false;
   }

   /**
    * Determines if the vehicle is done with the mission.
    *
    * @param   none
    * @return  flag indicating if the vehicle is done with the mission
    */
   public boolean isDone()
   {
      return LVMissionComplete;
   }


   /**
    * Get the AUV's current x position
    *
    * @param none
    * @return AUV's x position (meters)
    *
    */
   public double getCurrentXPosition()
   {
      return currentX;
   }
```

```
/**
 * Get the AUV's current y position
 *
 * @param none
 * @return AUV's y position (meters)
 *
 */
public double getCurrentYPosition()
{
   return currentZ;  //adjusting for VRML
}

/**
 * Get the AUV's current z position
 *
 * @param none
 * @return AUV's z position (meters)
 *
 */
public double getCurrentZPosition()
{
   return currentY;  //adjusting for VRML
}


/**
 * Get the AUV's current heading
 *
 * @param none
 * @return AUV's current heading (rads)
 *
 */
public double getCurrentHeading()
{
   return heading;
}

/**
 * Gets the current position for the AUV
 *
 * @param none
 * @return Array containing the position Index 0=x, 1=y, 2=z
 */
public double[] getCurrentPosition()
{
   double position[] = new double[3];
   position[0] = currentX;
   position[1] = currentY;
   position[2] = currentZ;
   return position;
}

/**
 * Outputs the leader's current position as a comma delimitted string
 *
 * @param none
 * @return comma delimitted string containing the leader's position
 *         in the format x, y, z
 */
public String getCurrentPositionAsString()
{
   return currentX + "," + currentY + "," + currentZ;
}
```

```
   /**
    * Gets the x-coordinate for the current waypoint
    *
    * @param   none
    * @return  the x-coordinate for the waypont currently being processed
(meters)
    */
   public double getCurrentWPX()
   {
         return XWay;
   }

   /**
    * Gets the y-coordinate for the current waypoint
    *
    * @param   none
    * @return  the y-coordinate for the waypont currently being processed
(meters)
    */
   public double getCurrentWPY()
   {
         return YWay;
   }

   /**
    * Gets the z-coordinate for the current waypoint.  Note that the current
    * version of the dynamics model does not include depth, but this method
    * is included to facilitate upgrades.
    *
    * @param   none
    * @return  the z-coordinate for the waypont currently being processed
(meters)
    */
   public double getCurrentWPZ()
   {
      return 0;
   }

   /**
    * Gets the waypoint currently being processed, i.e. the vehicle is
currently
    * transitting to this point
    *
    * @param   none
    * @return  array containing the current waypoint where index 0=X, 1=Y, 2=Z
    */
   public double[] getCurrentWaypoint()
   {
      double waypoint[] = new double[3];
      waypoint[0] = getCurrentWPX();
      waypoint[1] = getCurrentWPY();
      waypoint[2] = getCurrentWPZ();
      return waypoint;
   }

   /**
    * Gets the waypoint currently being processed as a comma delimited string
    *
    * @param   none
    * @return  string containing the current waypoint in the following format
    *          Waypoint X, Waypoint Y, Waypoint Z
    */
```

```
public String getCurrentWaypointAsString()
{
   return getCurrentWPX() + "," + getCurrentWPY() + "," + getCurrentWPZ();
}


/**
 * Gets the x-coordinate for the next waypoint to be processed
 *
 * @param   none
 * @return  the x-coordinate for the next waypoint to be processed (meters)
 */
public double getNextWPX()
{
   return 0;
}


/**
 * Gets the y-coordinate for the next waypoint to be processed
 *
 * @param   none
 * @return  the y-coordinate for the next waypoint to be processed (meters)
 */
public double getNextWPY()
{
   return 0;
}


/**
 * Gets the z-coordinate for the next waypoint to be processed
 * Note that the current version of the dynamics model ignores the z
 * component of waypoints, but this method is being included to facilitate
 * future upgrades.
 *
 * @param   none
 * @return  the z-coordinate for the next waypoint to be processed (meters)
 */
public double getNextWPZ()
{
   return 0;
}


/**
 * Gets the x,y, and z for the next waypoint to be processed
 *
 * @param   none
 * @return  array containing the next waypoint where index 0=X, 1=Y, 2=Z
 */
public double[] getNextWaypoint()
{
   double waypoint[] = new double[3];
   waypoint[0] = getNextWPX();
   waypoint[1] = getNextWPY();
   waypoint[2] = getNextWPZ();
   return waypoint;
}

/**
 * Gets the next waypoint to be processed (i.e. the waypoint that will be
 * processed after the vehicle reaches the "currentWaypoint".
```

```
     *
     * @param   none
     * @return  string containing the next waypoint in the following format
     *          Waypoint X, Waypoint Y, Waypoint Z
     */
    public String getNextWaypointAsString()
    {
       return getNextWPX() + "," + getNextWPY() + "," + getNextWPZ();
    }


    /**
     * Gets the the current and next waypoint information in string format.
     *
     * @param none
     * @return string containing the current and next waypoints in the
     *         format current WP x, current WP y, next WP x, next WP y
     */
    public String getWaypointsAsString()
    {
       return getCurrentWaypointAsString() + "," + getNextWaypointAsString();
    }

 /**
  * Generates the waypoints for ARIES based on information received using
  *
  * NOTE TO FUTURE DEVELOPERS - THIS IS THE FUNCTION TO MODIFY TO IMPLEMENT
  * NEW BEHAVIORS.
  *
  * @param   i = the current time increment (1/16 sec increments)
  * @param   j = the waypoint number (used for loitering)
  * @return  the next waypoint to be processed by ARIES
  */
    public double[] getNextWaypoint(int i, int j)
    {
       double[] returnValue = {0.0,0.0,0.0};
       boolean loiterFlag = false;   //flag indictating when a loiter is needed
       CommunicationsPacket LVData;//holds the LV data received via acomms
       RemusDynamics remusHandle = execSc.getPointerToRemus();

       //the following code is used to complete the LV's track
       //once the FV reaches the carrot, it will continue to reach the LV's true
 waypoint
       if(LVWPReached == false)
       {
          LVWPReached = true;
          returnValue[0] = LVCurrentWPX;
          returnValue[1] = LVCurrentWPY;
          returnValue[2] = 5;
          return returnValue;

       }

       if(CommunicationsPacket.getDeadZoneFlag() == false)
       {  //if not in a dead zone
          execSc.getPointerToReportGeneration().leaderMessageReceived();

          LVData = remusHandle.getRemusUpdate(LVWaypointCount++);
          if(LVData.getWaypointNumber() > LVWaypointCount)
          {  //if FV is more than one WP behind
             LVWaypointCount = LVData.getWaypointNumber();
             LVData = remusHandle.getRemusUpdate(LVWaypointCount);
          }
```

```
         LVSpeed = (double) LVData.getSpeed();
         LVHeading = (double) LVData.getHeading();
         LVMissionComplete = LVData.missionComplete();
         LVCourse = LVData.getCourseToWP();

         double[] tempPositionInfo;
         tempPositionInfo = LVData.getVehiclePosition();
         LVXPosition = tempPositionInfo[0];
         LVYPosition = tempPositionInfo[1];
         LVYPosition = tempPositionInfo[2];
         tempPositionInfo = LVData.getCurrentWaypoint();
         LVCurrentWPX = tempPositionInfo[0];
         LVCurrentWPY = tempPositionInfo[1];
         LVCurrentWPZ = tempPositionInfo[2];

         tempPositionInfo = LVData.getNextWaypoint();
         LVNextWPX = tempPositionInfo[0];
         LVNextWPY = tempPositionInfo[1];
         LVNextWPZ = tempPositionInfo[2];

         //calculate the time needed for the LV to reach the WP
         double LVXDist = StrictMath.pow((LVXPosition-LVCurrentWPX),2);
         double LVYDist = StrictMath.pow((LVYPosition-LVCurrentWPY),2);

         double LVDistToWP = StrictMath.sqrt(LVXDist + LVYDist);

         double LVTimeToWP = LVDistToWP/LVSpeed;

         double carrotDirection = LVCourse + StrictMath.PI;
            //the direction of the carrot relative to the WP currently being
            //processed by the LV

         returnValue[0] = LVCurrentWPX + carrot *
StrictMath.cos(carrotDirection);
         returnValue[1] = LVCurrentWPY +  carrot *
StrictMath.sin(carrotDirection);
         returnValue[2] = LVCurrentWPZ;

         double FVXDist = StrictMath.pow((getCurrentXPosition()-
returnValue[0]),2);
         double FVYDist = StrictMath.pow((getCurrentYPosition()-
returnValue[1]),2);

         double FVDistToWP = StrictMath.sqrt(FVXDist + FVYDist);

         double requiredSpeed = FVDistToWP/LVTimeToWP;

         if(j>1) //first wp after deployment is a special case
            setSpeed(requiredSpeed);
         currentlyLoitering = false;
         loiterPtCount = 0;
         LVWPReached = false;
      }
      if(CommunicationsPacket.getDeadZoneFlag() == true || loiterFlag == true)
      {
         execSc.getPointerToReportGeneration().leaderMessageLost();
         if(currentlyLoitering == false)
         {
            currentlyLoitering = true;
            loiterCenterX = X;
            loiterCenterY = Y;
            loiterPtCount = 0;
            setSpeed(DEFAULT_SPEED);
         }

         if(loiterPtCount == 0)
         {
            returnValue[0] = loiterCenterX;
            returnValue[1] = loiterCenterY-10.0;
            returnValue[2] = 5.0;
         }

         if(loiterPtCount == 1)
         {
            returnValue[0] = loiterCenterX+10.0;
            returnValue[1] = loiterCenterY;
            returnValue[2] = 5.0;
         }

         if(loiterPtCount == 2)
         {
            returnValue[0] = loiterCenterX;
            returnValue[1] = loiterCenterY+10.0;
            returnValue[2] = 5.0;
         }

         if(loiterPtCount == 3)
         {
            returnValue[0] = loiterCenterX-10.0;
            returnValue[1] = loiterCenterY;
            returnValue[2] = 5.0;
         }

         loiterPtCount++;  //inc loiter waypoint count
         if(loiterPtCount>3)
            loiterPtCount = 0;
      }

      returnValue[2] = 5;
      return returnValue;

   }

   /**
    * Sets ARIES' speed
    *
    * param newSpeed - new speed for ARIES in m/s
    * return void
    */
   public void setSpeed(double newSpeed)
   {
      final double maxSpeed = 2;
      final double minSpeed = 1;

      if(newSpeed<minSpeed)
      {
         newSpeed = minSpeed;
         System.out.println("ERROR - An attempt was made to set ARIES' speed
below min");
      }

      if(newSpeed>maxSpeed)
      {
         newSpeed = maxSpeed;
         System.out.println("ERROR - An attempt was made to set ARIES' speed
above max");
```

```
      }

      V = newSpeed;
      double comp2 = Yr - m*V;
      double[][] vals1 = {{Yv, comp2, 0},{Nv, Nr, 0},{0, 1, 0}};
      Matrix AA = new Matrix(vals1);


      U = newSpeed;
      RadCurv = U/(xss.get(0,0));
       SideSlip = StrictMath.atan2(xss.get(0,0),U)*180/StrictMath.PI;
   }

   /**
    * setup the insert point for the vehicle
    */
    public void setupInsert()
    {
      //calculate insert point for the FV
      RemusDynamics remusHandle = execSc.getPointerToRemus();

      double LVInsertX = remusHandle.getInsertPtX();
         //get the insert x position for REMUS

      double LVInsertY = remusHandle.getInsertPtY();
         //get the insert y position for remus

      //calculate the insert point for the LV including the carrot adjustment
      double carrotDir = remusHandle.getCourse() + StrictMath.PI;
      //make sure carrot dir is between 0 and 2pi
      while(carrotDir>= (2*StrictMath.PI))
         carrotDir -= (2*StrictMath.PI);

      double FVInsertX = LVInsertX + (carrot * StrictMath.cos(carrotDir));
      double FVInsertY = LVInsertY + (carrot * StrictMath.sin(carrotDir));

      PrevXWay = FVInsertX; //previously processed waypoint
      PrevYWay = FVInsertY;

      X = FVInsertX;        //current position of the vehicle
      Y = FVInsertY;

    }
}
```

LIST OF REFERENCES

Alleyne, J.C, *Position Estimation from Range Only Measurements*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 2000.

Ames, A.L., Nadeau, D.R., Moreland, J.L., *VRML 2.0 Sourcebook*, John Wiley & Sons, Inc., 2nd Ed., 1997.

Beckman, R., Martinez, A., Bourgeois, B., "AUV Positioning Using Bathymetry Matching," Oceans 2000 MTS/IEEE Conference and Exhibition, Providence, Rhode Island, September 2000.

Belcher, E.O., "REMUS with Sidescan Sonar and DIDSON," [http://www.apl.washington.edu/programs/DIDSON/Media/remus_with_didson.pdf], July 2003.

Benthos, "Telesonar Modems: The Solution to Wireless Underwater Communications," [http://www.benthos.com/pdf/Modems/telesonar.pdf], July 2003.

Bernstein, C., M. Connolly, M. Gavrilash, D. Kucik, S. Threatt, "Demonstration of Surf Zone Crawlers: Results from AUV Fest 01", Fifth Symposium on Technology and the Mine Problem, Monterey, California, April 2002.

Betke, M., Gurvits, L., "Mobile Robot Localization Using Landmarks," IEEE Transactions on Robotics and Automation, v. 12, no. 2, pp. 251 – 263, April 1997.

Brutzman, D.P., A Virtual World for an Autonomous Underwater Vehicle, PhD Dissertation, Naval Postgraduate School, Monterey, California, December 1994. Available at: http://web.nps.navy.mil/~brutzman/dissertation/

Burke, E.M., *Java and XSLT*, O'Reilly & Associates, Inc., 2001.

Carpin, S., Parker, L.E., "Cooperative Leader Following in a Distributed Multi-Robot System," Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington D.C., May 2002.

Chandak, P., *Study and Implementation of 'Follow the Leader'*, Master's Thesis, University of Cincinnati, Cincinnati, Ohio, 2002. Available at: http://reeses.mie.uc.edu/theses/theses2002/Thesis_Pravin.pdf

Chatfield, A.B., *Fundamentals of High Accuracy Inertial Navigation*, American Institute of Aeronautics and Astronautics, Inc., 1997.

Coastal Systems Station, *Lemming Acoustic Navigation System (LANS) Technical Note*, Tim McTrusty, unclassified, July 2000.

Coleman, J., “Undersea drones pull duty in Iraq hunting mines,” Cape Code Times, April 2, 2003, [http://www.hydroidinc.com/Hydroid_news/Cape_Cod_Times_Archives.pdf], accessed July 2003.

Dai, Y. P., Jin, C. Z.; Hu, J. L., Hirasawa, K., Liu, Z., “A Target Tracking Algorithm with Range Rate Under the Color Measurement Environment,” Proceedings of the Annual SICE Conference, pp 1145-1148, July 1999.

Dai, Y., Hirasawa, K., Murata, J., Hu, J., Jin, C., Liu, Z., “Use of Pseudo Measurement to Real-time Target Tracking,” IEEE System, Man, and Cybernetics 1999 Conference Proceedings, v. 5, pp. 33-38, October 1999.

Deffenbaugh, M., Bellingham, J., Schmidt, H., “The Relationship between Spherical and Hyperbolic Positioning,” Oceans 1996 MTS/IEEE: Prospects for the 21st Century, v. 2, pp 590-595, September 1996.

Deitel, H.M., Deitel, P.J., Nieto, T.R., Lin, T.M., Sadhu, P., *XML: How to Program*, Prentice Hall, 2001.

Du, Y., Papanikolopoulos, N.P., “Real-Time Vehicle Following through a Novel Symmetry-Based Approach,” Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, April 1997.

Gans, R.F., “A control algorithm for automated pursuit,” Proceedings of the 1997 International Conference on Control Applications, Hartford, Connecticut, October 1997.

Grewal, M.S., Weill, L.R., Andrews, A.P., *Global Positioning Systems, Inertial Navigation, and Integration*, John Wiley & Sons, Inc., 2001.

Gruneisen, A., Henriet, Y., *3D Model of the ARIES AUV JavaDoc for Dynamics AUV Mission Visualization Workbench AUV Dynamics Control Workbench on Matlab*, Naval Postgraduate School, NPS-ME-02-005, 2002.

Healey, A. J. Marco, D. B., "Command, Control and Navigation: Experimental Results with the NPS ARIES AUV," IEEE Journal of Oceanic Engineering, Special Issue on Autonomous Ocean Sampling Networks, vol.26, n.4, October 2001.

Healey, A. J., “Dynamics and Control of Mobile Robotic Vehicles,” [http://web.nps.navy.mil/~me/healey/ME4823/Ch1_2.doc], January 2003.

Healey, A. J., Marco, D. B., "Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV," Proceedings of IEEE Oceans 2000, Providence, RI, September 2000.

Hogg, R.W., and others, “Algorithms and Sensors for Small Robot Path Following,” Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington D.C., May 2002.

Hunter, D., *Beginning XML*, Wrox Press Ltd., 2000.

Hydroid Inc., “Discover REMUS,” [http://www.hydroidinc.com/remus.htm], June 2003.

Kearfott Guidance & Navigation Corporation, “Seaborne Inertial Navigation System Doppler Velocity Log (SEADeViL),” KN-6050 Family Technical Data Sheet, July 2001.

Kehtarnavaz, N., and others, “A Transportable Neural-Network Approach to Autonomous Vehicle Following,” IEEE Transactions on Vehicular Technology, Vol. 47, No. 2, May 1998.

Krantz, D., Gini, M., “Non-Uniform Dead-Reckoning Position Estimate Updates,” Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, Minnesota, April 1996.

Kucik, D.P., *Self-Contained, Self-Surveying Differential GPS Base Station and Method of Operating Same,* U.S. Patent 6,380,888, 30 April 2002.

Larsen, M.B., “Synthetic Long Baseline Navigation of Underwater Vehicles,” Oceans 2000 MTS/IEEE Conference and Exhibition, v. 3, pp 2043-2050, September 2000.

Maloney, E.S., *Dutton’s Navigation and Piloting*, 14th ed., Naval Institute Press, 1985.

Marr, W.J., Acoustic Based Tactical Control of Underwater Vehicles, PhD Dissertation, Naval Postgraduate School, Monterey, California, June 2003.

Matos, A., Cruz, N., Martins, A., Pereira, F., “Development and Implementation of a Low-Cost LBL Navigation System for an AUV,” Oceans 1999 MTS/IEEE: Riding the Crest into the 21st Century, v. 2, pp774-779, September 1999.

Muralidharan, A., *Sonar Based Navigation: Follow the Leader Solution for Bearcat III*, Master’s Thesis, University of Cincinnati, Cincinnati, Ohio, 2001.

Nguyen, T.V., *ARIES Navigation System Accuracy and Track Following*, Master’s Thesis, Naval Postgraduate School, Monterey, California, March 2002.

NPS Center for AUV Research, “About the ARIES AUV,” [http://www.cs.nps.navy.mil/research/auv/auvstats.html], July 2003.

NPS Center for AUV Research, “ARIES Azores 01 Photos,” [http://www.cs.nps.navy.mil/research/auv/images/aries_thumbnail.html], July 2003.

Pilbrow, E.N., Hayes, M.P., Gough, P.T., “Long Baseline Precision Navigation System for Synthetic Aperture Sonar,” [http://www.elec.canterbury.ac.nz/research/acoustics/Papers/Pilbrow_arspc02.pdf], August 2003.

Ransford, G.A., Ioup, J.W., “Locating and Determining the Orientation of Underwater Research Equipment: Acoustic Range and Range Rate Data,” IEEE Journal of Oceanic Engineering, v. 12, no. 3, pp. 524-534, July 1987.

Reimers, S.P., *Towards Internet Protocol Over Seawater (IP/SW): Forward Error Correction (FEC) Using Hamming Codes for Reliable Acoustic Telemetry*, Master’s Thesis, Naval Postgraduate School, Monterey, California, September 1995.

Tamura, K., Furukawa, Y., “Autonomous Vehicle Control System of ICVS City Pal: Electric Tow-bar function,” Proceedings of the IEEE Intelligent Vehicles Symposium 2000, Dearborn, Michigan, October 2000.

Tervalon, N., Kirkwood, W., “Ice Profiling Sonar for an AUV: An approach for obtaining SCICEX quality ice draft data,” [http://www.aslenv.com/reports/OIA%202001%20MBARI%20Tervalon%20Paper.pdf], July 2003.

Titerton, D.H., Weston, J.L., *Strapdown Inertial Navigation Technology*, Peter Peregrinus Ltd., 1997.

University of Pennsylvania Naval Reserve Officer Training Corps, “Navigation Training: Electronic Navigation,” [http://www.upenn.edu/nrotc/ns301/lesson20.pdf], accessed June 3, 2003.

von Alt, C., “Autonomous Underwater Vehicles,” *Autonomous Underwater Lagrangian Platforms and Sensors Workshop*, March 2003, [http://www.geo-prose.com/ALPS/white_papers/alt.pdf], accessed July 2003.

W3C, “Extensible Markup Language (XML),” [http://www.w3.org/XML/], August 2003.

Web3D, “Extensible 3D (X3D) Graphics,” [http://www.web3d.org/x3d.html], August 2003.

Web3D, “Extensible 3D (X3D) Graphics: X3D Task Group Source Code and Applications,” [http://www.web3d.org/TaskGroups/x3d/x3d_scapps.html], accessed October 2003.

Wikipedia, “Triangulation,” [http://www.wikipedia.org/wiki/Triangulation], August 2003.

Williams, D.L., *Loitering behavior of Autonomous Underwater Vehicles*, Master’s Thesis, Naval Postgraduate School, Monterey, California, June 2002.

Xj3D, “The Xj3D Project,” [http://www.xj3d.org], August 2003.

Yamakita, M., Suh, J., “Adaptive Generation of Desired Velocity Field for Leader-Follower Type Cooperative Mobile Robots with Decentralized PVFC,” Proceedings of the 2001 IEEE International Conference on Robotics & Automation, Seoul, Korea, May 21-26 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. RADM Mike Tracy
   Submarine Warfare Division (N77)
   Washington, DC

4. RADM William Timme
   Newport, Rhode Island
   Commander, Naval Undersea Warfare Center

5. RADM John Pearson, USN (Ret)
   Naval Postgraduate School
   Monterey, California

6. CAPT Anthony Shutt
   Commanding Officer, Naval Surface Warfare Center – Panama City
   Panama City, Florida

7. CAPT John Mickey
   Commander, Naval Undersea Warfare Center Newport Division
   Newport, Rhode Island

8. Dr. David Skinner
   Product Area Director, Naval Surface Warfare Center – Panama City
   Panama City, Florida

9. Don Brutzman
   Naval Postgraduate School
   Monterey, California

10. Tony Healey
    Naval Postgraduate School
    Monterey, California

11. Doug Horner
    Naval Postgraduate School
    Monterey, California

12. Dr. Doug Todoroff
Office of Naval Research
Arlington, Virginia

13. Phil Bernstein
Naval Surface Warfare Center – Panama City
Panama City, Florida

14. Bette Bruhmuller
Naval Surface Warfare Center – Panama City
Panama City, Florida

15. Dick Blidberg
Director, Autonomous Undersea Systems Institute (AUSI)
Lee, New Hampshire